



Hi309a 通用 MPU 芯片

用户指南 02-加速引擎

文档版本 02

发布日期 2025-12-12

版权所有 © 海思技术有限公司2026。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

海思技术有限公司

地址：上海市青浦区虹桥港路2号101室 邮编：201721

网址：<https://www.hisilicon.com/cn/>

客户服务邮箱：support@hisilicon.com



前言

概述

本文档主要介绍Hi309a芯片加速引擎的基本功能，并提供了常见应用模式下的配置方法。

读者对象

本文档主要适用于以下工程师：

- 技术支持工程师
- 单板硬件开发工程师
- 单板软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。



修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2025-10-30	01	第1次正式版本发布。
2025-12-12	02	第2次正式版本发布。

手册使用约束

须知

在使用Hi309a芯片时，请严格按照芯片手册中注明的功能、性能、配置步骤等信息进行软硬件开发。若未按手册注明的信息进行操作，可能会出现不可预知的风险。任何不符合手册注明的应用需求，必须获得芯片SE（系统工程师）或者FAE（现场应用工程师）的书面批准后方可实施。请勿根据以往商用器件的经验进行推测，否则一切后果自行承担，感谢您的理解与配合。



目录

前言.....	i
1 BMU.....	1
1.1 概述.....	1
1.2 关键术语.....	1
1.3 特性.....	2
1.4 遵循协议.....	7
1.5 功能描述.....	7
1.5.1 典型应用.....	7
1.5.1.1 应用框图.....	8
1.5.1.2 模块框图.....	9
1.5.1.3 申请 Buffer.....	10
1.5.1.4 释放 Buffer.....	12
1.5.1.4.1 释放链式 Buffer.....	13
1.5.1.4.2 释放分离式 Buffer.....	13
1.5.1.5 物理 Pool 动态注入 Block.....	14
1.5.1.6 物理 Pool 动态删除 Block.....	14
1.5.1.7 软件管理模式.....	14
1.5.1.8 QoS 反压.....	15
1.5.2 多模及虚拟化.....	16
1.5.3 缓存跟踪.....	25
1.5.4 硬件魔术字.....	26
1.5.5 内部时间戳.....	26
1.5.6 预取 FIFO.....	27
1.6 编程接口.....	27
1.6.1 Buffer 指针结构.....	27
1.6.2 注入 block 数据结构.....	28
1.6.3 维测信息.....	28
1.6.4 性能和功耗配置.....	29
1.6.4.1 性能优化配置.....	29
1.6.4.1.1 链式 Buffer 相关功能配置约束.....	29
1.6.4.1.2 寄存器 BMU_POOL_BLOCK_NUM_P 的配置约束.....	30
1.7 应用配置.....	31
1.7.1 软件指南.....	31



1.7.1.1 Buffer 碎片化回收.....	31
1.7.1.2 BMU PF 初始化.....	31
1.7.1.3 VF 业务初始化.....	33
1.7.2 VF 内存迁移流程.....	34
1.7.3 BMU 多模复位的方法.....	35
1.7.4 中断和中断查询.....	36
1.8 性能.....	37
1.9 DFX 应用指导.....	37
1.9.1 故障在线检测.....	38
1.9.1.1 BMU 配置异常.....	38
1.9.1.2 典型问题定位.....	38
1.9.1.2.1 申请 Buffer 异常.....	38
1.9.1.2.2 释放 Buffer 异常.....	38
1.9.1.2.3 链式 Buffer 释放时 PPE 发送方向描述符 OWN 域不为 0xA 中断.....	38
1.9.1.2.4 软件管理模式 Buffer 真满丢弃.....	39
1.9.1.2.5 软件管理模式假满的反压.....	39
1.9.1.2.6 BMU 魔术字校验失败.....	39
1.9.1.2.7 链式 Buffer 循环链的节点异常.....	39
1.9.1.2.8 链式 Buffer 或分离式 Buffer 受到内部队列假满的反压.....	39
1.9.1.2.9 逻辑 pool 被读空中断.....	39
1.9.1.2.10 逻辑 pool 的内部 SRAM 读空中断.....	39
1.9.1.2.11 逻辑 POOL 的 Buffer 剩余低于配置的所有 9 个水线.....	39
1.9.2 模块维测.....	39
1.9.2.1 内部状态查询.....	40
1.9.2.2 统计.....	40
1.9.2.2.1 功能统计.....	40
1.9.2.3 CPU 间接访问内部表项 RAM.....	42
1.9.3 典型定位案例.....	43
1.10 注意事项.....	43
2 TM.....	44
2.1 概述.....	44
2.2 关键术语.....	44
2.3 特性.....	45
2.4 遵循协议.....	51
2.5 功能描述.....	51
2.5.1 应用框图.....	51
2.5.2 相关模块.....	52
2.5.3 模块框图.....	53
2.5.4 数据结构.....	53
2.5.5 包描述符添加.....	54
2.5.6 队列管理.....	54
2.5.7 拥塞避免.....	55



2.5.8 队列调度.....	59
2.5.9 流量整形.....	64
2.5.10 Port Trunk.....	67
2.5.11 紧急队列.....	68
2.5.12 反压处理.....	68
2.5.13 多模隔离.....	69
2.5.14 中断上报.....	75
2.5.15 统计功能.....	76
2.5.16 DFX.....	76
2.5.17 时戳.....	76
2.5.18 动态时钟门控.....	77
2.5.19 Telemetry 捕获功能.....	77
2.6 软件指南.....	78
2.6.1 PF 初始化.....	78
2.6.2 TM VF 初始化.....	81
2.6.3 TM 节点链接关系动态配置.....	82
2.6.4 TM 节点 Shaping 参数动态配置.....	84
2.6.5 TM 端口超时丢弃步骤.....	84
2.6.6 TM 多模复位不释放 Buffer 步骤.....	84
2.6.7 TM 多模复位释放 Buffer 步骤.....	85
2.6.8 TM 安全复位不释放 Buffer 步骤.....	85
2.6.9 TM 安全复位释放 Buffer 步骤.....	86
2.7 性能.....	87
2.7.1 应用场景.....	87
2.7.2 实际测试结果.....	87
2.7.3 吞吐率 (Throughput)	87
2.7.4 延迟 (Latency)	88
3 POE.....	89
3.1 概述.....	89
3.2 关键术语.....	89
3.3 特性.....	91
3.4 遵循协议.....	97
3.5 功能描述.....	97
3.5.1 POE 应用.....	97
3.5.1.1 相关模块.....	99
3.5.1.2 模块框图.....	101
3.5.2 CPU 访问寄存器.....	102
3.5.3 CPU 加包 (ADD_PACKET)	102
3.5.4 CPU 添加保序命令.....	102
3.5.4.1 SWITCH_NULL.....	103
3.5.4.2 SWITCH_DESCCHEDULE.....	103
3.5.4.3 DESCCHEDULE.....	103



3.5.4.4 SEND_TM/SEND_POE.....	103
3.5.5 CPU 添加转发 TM 任务.....	104
3.5.6 加速器加包 (ADD_PACKET)	105
3.5.7 CPU 获取包.....	106
3.5.7.1 命令类型.....	106
3.5.7.2 命令处理.....	107
3.5.8 加速器获取包.....	108
3.5.9 入口反压功能.....	108
3.5.10 出口反压功能.....	108
3.5.11 统一的加速器访问接口.....	108
3.5.11.1 与所有加速器一致的中断地址和中断编号.....	108
3.5.11.2 软件可查询硬件规格信息.....	108
3.5.11.3 软件可读写设备号 PF_ID, 功能号 VF_ID.....	109
3.5.11.4 采用统一的虚拟化 PF/VF 接口.....	109
3.5.12 支持多模及资源的虚拟化.....	109
3.5.12.1 Group 资源的虚拟化.....	109
3.5.12.2 逻辑 CPU 资源的虚拟化.....	109
3.5.12.3 寄存器资源的虚拟化.....	109
3.5.13 任务的缓存.....	110
3.5.13.1 新任务缓存.....	110
3.5.13.2 Atomic 流拥塞.....	112
3.5.13.3 CPU 解绑定任务缓存.....	114
3.5.13.4 CPU 流分类任务缓存.....	115
3.5.14 加速器任务处理通知.....	115
3.5.15 任务负载均衡.....	116
3.5.15.1 Group 的划分.....	116
3.5.15.2 负载均衡配置.....	116
3.5.16 任务 QoS 调度管理.....	118
3.5.16.1 逻辑 CPU 的任务调度.....	118
3.5.16.2 加速器的任务调度.....	119
3.5.16.3 调度场景说明.....	120
3.5.17 包任务保序 (多核并发处理同流)	121
3.5.17.1 保序队列.....	123
3.5.17.2 保序命令处理.....	124
3.5.18 包任务通讯.....	127
3.5.19 核间消息通讯.....	127
3.5.20 WFI/WFE 唤醒 CPU 读取任务.....	127
3.5.20.1 事件触发通知 (WFE)	127
3.5.20.2 中断通知 (WFI)	128
3.5.21 多模隔离.....	128
3.5.22 模式单独复位.....	131
3.5.23 推送包到 Cacheable 的存储空间.....	132



3.5.24 从 Cacheable 的存储空间获取包.....	133
3.5.25 CPU 正常业务中断.....	134
3.5.25.1 物理队列收包中断.....	134
3.5.25.2 Group 粒度超时未取包中断.....	134
3.5.25.3 Group 粒度水线中断.....	135
3.5.25.4 物理队列高低水线中断.....	135
3.5.26 原子特性.....	135
3.6 软件指南.....	136
3.6.1 大小端转换.....	136
3.6.2 数据结构.....	138
3.6.3 寄存器动态配置.....	138
3.6.4 POE 初始化.....	139
3.6.4.1 模块级初始化.....	139
3.6.4.2 业务级初始化.....	142
3.6.4.3 独立软复位.....	144
3.6.4.4 模式独立复位.....	145
3.7 性能.....	146
3.7.1 吞吐率 (Throughput)	146
3.7.2 延迟 (Latency)	146
3.8 DFX 应用指导.....	146
3.8.1 可靠性.....	146
3.8.1.1 CPU 加包防止错误传播.....	147
3.8.1.2 保序队列队头超时异常处理.....	147
3.8.1.3 CPU 加包异常.....	147
3.8.1.4 加速器加包异常.....	149
3.8.1.5 物理队列的满中断.....	149
3.8.1.6 模式粒度的水线中断.....	149
3.8.1.7 接口命令错误.....	149
3.8.1.8 内部数据完整性错误.....	149
3.8.2 模块维侧.....	150
3.8.2.1 Debug.....	150
3.8.2.2 POE 内部 RAM 软件可访问.....	150
3.8.2.3 ECC 错误注入.....	150
3.8.2.4 ECC 校验功能的打开关闭.....	150
3.8.2.5 中断注入.....	151
3.8.2.6 时戳功能.....	151
3.9 注意事项.....	151
3.9.1 SWITCH_NULL 命令的使用注意事项.....	151
4 PPE.....	152
4.1 关键术语.....	152
4.2 概述.....	153
4.3 特性.....	154



4.4 功能描述	162
4.4.1 相关模块	162
4.4.2 PPE 通道和网口互联状况和使用场景	162
4.4.2.1 PPE_TNL 和 MAC_TNL	163
4.4.3 典型数据流	165
4.4.3.1 发送直通数据流	165
4.4.3.2 接收直通数据流	166
4.4.3.3 LSW 数据流	166
4.4.3.3.1 MAC 间转发	167
4.4.3.3.2 CPU 间数据流	168
4.4.3.3.3 MAC 到 CPU 数据流	169
4.4.3.3.4 CPU 到 MAC 数据流	170
4.4.4 常用业务描述	170
4.4.4.1 调试网口功能	170
4.4.4.2 RTC 功能	171
4.4.4.3 1731 功能	171
4.4.4.3.1 发送方向	171
4.4.4.4 1588 功能	171
4.4.4.4.1 1588 opcode	171
4.4.4.4.2 接收方向处理	173
4.4.4.4.3 发送方向处理	173
4.4.4.4.4 时戳恢复算法	175
4.4.4.5 LSW 和直通分发的 QoS 处理	175
4.4.4.5.1 入口 CoS 生成	175
4.4.4.5.2 出口 TC 生成	176
4.4.4.6 ROCE 报文处理	177
4.4.4.7 通用报文解析	178
4.4.4.8 自定义报文解析	178
4.4.4.9 畸形报文的检查	179
4.4.5 LSW 转发和 CPU 分发	180
4.4.5.1 直通表	180
4.4.5.2 CPU 分发	181
4.4.5.2.1 分发概述	182
4.4.5.2.2 分发规则	183
4.4.5.2.3 QoS 生成	194
4.4.5.3 LSW 转发	196
4.4.5.3.1 LSW VLAN 特性	196
4.4.5.3.2 流表	206
4.4.5.3.2.1 流表的 key 选择	207
4.4.5.3.2.2 入口流表	208
4.4.5.3.2.3 出口流表	219
4.4.5.3.3 L2 表	223



4.4.5.3.3.1 L2 表概述.....	223
4.4.5.3.4 老化.....	229
4.4.5.3.5 生成树.....	231
4.4.5.3.6 Trunk (链路聚合)	232
4.4.5.3.7 镜像.....	238
4.4.5.3.8 VLAN 过滤表.....	239
4.4.5.3.9 多播复制表.....	239
4.4.5.3.10 未知报文.....	240
4.4.5.3.11 BPDU 报文.....	240
4.4.5.3.12 LSW 直通报文.....	240
4.4.5.3.13 端口隔离.....	240
4.4.5.3.14 转发流程.....	241
4.4.5.3.15 VLAN 卸载流程.....	247
4.4.5.3.16 镜像流程.....	248
4.4.5.4 RSS 功能.....	249
4.4.6 内部报文缓存管理.....	250
4.4.6.1 流控功能.....	250
4.4.6.1.1 普通流控.....	250
4.4.6.2 队列管理.....	250
4.4.6.3 典型场景 Buffer 和水线配置.....	251
4.4.6.3.1 多播 FIFO 配置.....	251
4.4.6.3.2 队列配置.....	251
4.4.7 BMU、POE 交互.....	251
4.4.7.1 BMU Buffer 申请与释放.....	251
4.4.7.2 BMU QoS 丢弃报文.....	252
4.4.7.3 POE 消息交互.....	252
4.4.8 接收通路.....	253
4.4.8.1 支持的接收报文存储方式.....	253
4.4.8.2 Write allocate 功能.....	254
4.4.9 发送通路.....	254
4.4.9.1 支持的发送报文存储方式.....	254
4.4.9.2 报文长度约束.....	255
4.4.10 Telemetry 功能.....	255
4.4.10.1 INT/IOAM 业务流程.....	255
4.4.10.2 报文接收.....	256
4.4.10.3 发送方向.....	259
4.4.11 多模隔离.....	260
4.4.12 典型报文处理方式.....	261
4.4.12.1 Checksum 处理.....	261
4.4.12.1.1 发送侧 Checksum 计算.....	262
4.4.12.1.2 接收端 Checksum 校验.....	263
4.4.12.1.3 UDP Checksum 为 0 的处理.....	263



4.4.12.1.4 异常报文的 Checksum.....	264
4.4.13 临终遗言报文.....	265
4.5 编程接口.....	266
4.5.1 数据结构.....	266
4.6 应用配置.....	278
4.6.1 报文分发功能.....	278
4.6.2 测试网口功能.....	278
4.6.2.1 接收方向.....	278
4.6.2.2 发送方向.....	278
4.6.3 各个表项配置流程.....	279
4.6.3.1 直通表.....	279
4.6.3.2 TCAM 配置.....	279
4.6.3.3 L2 表配置.....	280
4.6.4 ETS 配置.....	282
4.6.4.1 调度结构图.....	282
4.6.4.2 配置寄存器.....	283
4.6.4.3 链接关系配置举例.....	286
4.6.4.4 调度权重配置举例.....	286
4.6.4.5 Shaping 配置.....	286
4.7 复位和初始化.....	288
4.7.1 强制复位.....	288
4.7.2 安全复位/PF 复位.....	289
4.7.3 VF 复位.....	291
4.7.4 初始化.....	292
4.7.4.1 PPE 复位后初始化流程.....	292
4.7.4.2 PA 寄存器配置流程.....	294
4.7.4.3 TPU 寄存器配置流程.....	295
4.7.4.4 MAM 初始化流程.....	295
4.7.4.5 RPU 初始化流程.....	296
4.7.4.6 PPP 初始化流程.....	296
4.7.4.7 SSU 初始化流程.....	297
4.7.5 LSW alone.....	300
4.8 芯片调试 DFX 功能特性描述.....	300
4.8.1 可靠性.....	300
4.8.2 环回.....	300
4.8.3 模块维测.....	301
4.8.3.1 统计.....	301
4.8.3.2 告警上报.....	302
4.8.3.3 中断上报.....	302
4.8.3.4 消息上报.....	303
4.8.3.5 Debug.....	303
4.8.3.6 注意事项.....	303



4.9 故障处理指南.....	303
4.9.1 软硬件接口故障.....	303
4.9.2 硬件内部数据完整性错误.....	304
4.10 典型问题定位流程.....	304
4.11 时钟复位.....	304
4.12 PPE_TNL 的性能指标.....	304
4.12.1 单 TNL 单 BD 性能.....	304
4.12.2 单 TNL 多 BD 性能.....	304
4.12.3 多 TNL 单 BD 性能.....	304
4.12.3.1 转发性能.....	305
4.12.4 吞吐率 (Throughput)	305
4.12.5 延迟 (Latency)	305
5 SA 加速器.....	306
5.1 概述.....	306
5.2 关键术语.....	306
5.3 特性.....	307
5.4 遵循协议.....	309
5.5 功能描述.....	309
5.5.1 SA 加速器应用图.....	309
5.5.2 字符串匹配与正则匹配.....	310
5.5.3 块模式匹配和流模式匹配.....	310
5.5.4 模式匹配结果上报处理.....	311
5.5.5 知识库加载功能.....	311
5.5.6 SA 获取 task (GET_POE)	311
5.5.7 SA 添加 task (ADD_PACKET)	311
5.5.8 连续字符丢弃 (8+8) 功能.....	311
5.5.9 结果去重功能.....	312
5.6 编程接口.....	313
5.6.1 SA 加速器数据结构.....	313
5.6.2 大小端转换.....	327
5.6.3 知识库加载和回读接口.....	328
5.7 应用配置.....	328
5.8 初始化.....	328
5.8.1 知识库加载和回读接口.....	328
5.8.2 硬件引擎全局配置.....	328
5.8.3 知识库编译.....	330
5.8.4 知识库加载.....	330
5.8.4.1 知识库编译结果.....	330
5.8.4.2 加载流程.....	330
5.8.5 业务开工配置.....	334
5.8.6 FE 中断 AXI 总线关闭后打开流程.....	335
5.8.7 软复位.....	335



5.9 DFX 应用指导.....	336
5.9.1 故障在线检测.....	336
5.9.1.1 SA 配置异常.....	337
5.9.1.2 典型问题定位.....	338
5.9.1.2.1 SA 加速器不获取任务不开工或无反应.....	338
5.9.1.2.2 PPM 模块出现“pkt_rd 读取 buf 类型报文时其长度总和和 BD 长度域不一致错误”.....	339
5.9.1.2.3 SA 出现“AXI 总线响应 resp 异常中断”.....	339
5.9.1.2.4 SA 出现下发匹配任务后不返回响应消息（ADD_PACKET）.....	340
5.9.1.2.5 SA 出现 BD 内容 CRC 校验错误或报文内容 CRC 校验错误.....	340
5.9.1.2.6 SA 出现知识库加载超时中断.....	341
5.9.1.2.7 CPM 模块出现流表基地址无效异常中断.....	341
5.9.1.2.8 CPM 模块出现流表数据奇偶校验异常中断.....	341
5.9.1.2.9 CPM 模块出现流表队列一致性异常中断.....	341
5.9.1.2.10 分组匹配结果有丢失现象一.....	341
5.9.1.2.11 分组匹配结果有丢失现象二.....	342
5.9.1.2.12 分组匹配结果有丢失现象三.....	342
5.9.1.2.13 无分组命中结果现象.....	342
5.9.1.2.14 匹配结果有丢失现象.....	342
5.9.1.2.15 匹配结果中有重复的命中项.....	342
5.9.1.2.16 所有报文均上报无命中的匹配结果（预期有命中）.....	342
5.9.1.2.17 匹配结果中缺失部分预期的命中项.....	342
5.9.2 模块维测.....	343
5.9.2.1 内部状态查询.....	343
5.9.2.2 统计.....	347
5.9.2.2.1 功能统计.....	347
5.9.2.2.2 性能统计.....	355
5.9.3 典型定位案例.....	355
5.10 注意事项.....	356
6 SA 规则转换器.....	357
6.1 概述.....	357
6.2 关键术语.....	357
6.3 特性.....	358
6.4 遵循协议.....	358
6.5 功能描述.....	360
6.5.1 SA 加速器应用图.....	361
6.5.2 SA 转换器处理过程.....	362
6.6 编程接口.....	363
6.6.1 规则信息结构体.....	363
6.6.2 编译结果结构体.....	363
6.6.3 转换器接口函数定义.....	364
6.6.3.1 CheckGrammer.....	364
6.6.3.2 OpenConverter.....	364



6.6.3.3 CloseConverter.....	364
6.6.3.4 AddRule.....	365
6.6.3.5 Convert.....	365
6.6.3.6 Combine.....	365
6.6.3.7 GetResult.....	365
6.6.3.8 FreeResult.....	366
6.6.3.9 RegisterAllocFree.....	366
6.6.3.10 RegisterLogFunc.....	366
6.6.3.11 RegisterDbgFunc.....	366
6.6.3.12 RegisterLinkListDepth.....	367
6.6.3.13 RegisterKeyModel.....	367
6.6.3.14 RegisterBadkeyPath.....	367
6.6.3.15 SetDfaThreshold.....	367
6.6.3.16 SetLinkListWidth.....	368
6.6.3.17 SetResourceLimit.....	368
6.6.3.18 RegisterMemChkFunc.....	368
6.6.3.19 GetDataCrc.....	368
6.6.3.20 GetResultCrc.....	369
6.6.3.21 SetOptimizeToggle.....	369
6.7 应用配置.....	370
6.8 返回码说明.....	371
7 TRING.....	373
7.1 概述.....	373
7.2 关键术语.....	374
7.3 特性.....	374
7.4 遵循协议.....	375
7.5 功能描述.....	375
7.5.1 典型应用.....	376
7.5.2 TRING 功能原理.....	376
7.5.3 多模隔离.....	378
7.5.4 原子特性.....	378
7.6 软件指南.....	378
7.6.1 PF 初始化.....	378
7.6.2 软复位.....	379
7.6.3 VF 卸载.....	379
7.6.4 时钟门控.....	379
7.6.5 关闭时钟.....	380
7.6.6 VF 初始化.....	380
7.6.7 Create 一个定时器.....	380
7.6.8 Start 一个定时器.....	380
7.6.9 Resume 一个定时器.....	381
7.6.10 Pause 一个定时器.....	381



7.6.11 Stop 一个定时器.....	381
7.6.12 Delete 一个.....	382
7.6.13 TRING 挂接定时器操作.....	382
7.6.14 TRING 解挂定时器操作.....	382
7.6.15 处理定时器 WHEEL 操作.....	383
7.6.16 处理定时器操作.....	383
7.7 性能.....	383
7.7.1 CPU 添加任务处理延迟分析.....	384
7.7.2 发送 POE 消息吞吐量.....	385
7.8 DFX 应用指导.....	386
7.8.1 故障在线检测.....	386
7.8.1.1 TRING 配置异常.....	386
7.8.1.1.1 虚拟 wheel 与物理 wheel 不匹配.....	386
7.8.1.1.2 虚拟 CPU 与物理 CPU 不匹配.....	386
7.8.2 维测统计.....	386
7.8.3 典型定位案例.....	386
7.9 注意事项.....	386
8 SEC.....	388
8.1 概述.....	388
8.2 关键术语.....	389
8.3 特性.....	390
8.4 遵循协议.....	396
8.5 功能描述.....	397
8.5.1 IPsec 机密性和完整性协议处理.....	397
8.5.1.1 IPsec ESP 协议处理.....	403
8.5.1.2 IPsec AH 协议处理.....	405
8.5.2 OpenSSL/TLS1.2 机密性和完整性协议功能.....	407
8.5.2.1 OpenSSL/TLS1.2/DTLS 报文格式.....	407
8.5.2.2 OpenSSL/TLS 软件配置.....	407
8.5.3 TLS 1.3 机密性和完整性协议处理.....	413
8.5.3.1 SEC 支持的功能介绍.....	413
8.5.3.2 加密方向的软硬件配合.....	414
8.5.3.3 解密方向的软件配置.....	414
8.5.3.4 TLS1.3 GCM 流模式.....	415
8.5.4 流处理模式加解密功能.....	415
8.5.5 长数据 Hash 功能.....	418
8.5.6 多模隔离.....	419
8.5.7 SEC 大小端翻转功能.....	420
8.5.8 IVIN 的获取.....	421
8.5.9 UDP checksum 域计算.....	423
8.5.9.1 补码算法.....	423
8.5.9.2 UDP Header 介绍.....	423



8.5.9.3 IPv6 Header 介绍.....	424
8.5.9.4 IPv6 + UDP 应用.....	424
8.5.9.5 Check Sum 计算.....	425
8.5.9.6 Check Sum 注意事项.....	426
8.5.10 密钥管理.....	426
8.5.10.1 密钥类型.....	426
8.5.10.2 功能特性.....	427
8.5.10.3 DDR 中密文 KEY 的数据格式.....	427
8.5.11 一级 UNWRAP.....	427
8.5.11.1 一级 UNWRAP 操作.....	428
8.5.12 低功耗.....	428
8.5.13 透传包处理.....	428
8.5.14 BD IVLD.....	428
8.6 编程接口.....	428
8.6.1 SEC BD (SQE)	429
8.6.2 SEC 输入包任务描述符.....	462
8.6.3 SEC 输出包任务描述符.....	463
8.6.4 数据访问方式.....	468
8.6.4.1 Pbuffer 形式访问.....	468
8.6.4.2 SGL 形式访问.....	468
8.6.4.3 PPE List 形式访问.....	470
8.6.5 软硬件接口.....	471
8.6.5.1 POE 接口工作方式.....	471
8.6.5.2 长数据 HASH 操作流程.....	472
8.6.5.3 BD FIFO (安全任务) 接口工作方式.....	472
8.6.5.4 BD FIFO 工作方式资源回收.....	473
8.6.5.5 单模资源回收和丢弃流程.....	473
8.6.5.6 单模内部模块复位流程.....	474
8.6.5.7 单模式初始化流程.....	474
8.6.6 BD FIFO 工作方式任务控制.....	475
8.6.7 性能和功耗配置.....	478
8.6.7.1 时钟门控.....	479
8.6.8 固件编程接口.....	479
8.6.9 AEAD 算法描述符配置.....	479
8.6.9.1 AES-CCM 算法描述符配置.....	479
8.6.9.2 AES-GCM 算法描述符配置.....	481
8.7 初始化.....	481
8.7.1 软复位.....	481
8.7.2 初始化.....	483
8.7.2.1 安全启动配置.....	483
8.7.2.2 公共初始化.....	483
8.7.2.3 BD FIFO 接口初始化.....	484



8.7.2.4 POE 接口初始化.....	485
8.7.2.5 SEC POE 工作方式下的 VF 业务相关初始化.....	486
8.7.2.6 KM 秘钥加载.....	487
8.7.3 软复位.....	487
8.8 DFX 应用指导.....	489
8.8.1 SEC 常见异常处理.....	489
8.8.2 PD MEM 和 BD MEM 可视化.....	554
8.8.3 典型定位案例.....	554
8.9 注意事项.....	555
9 TRNG.....	556
9.1 概述.....	556
9.2 关键术语.....	557
9.3 特性.....	558
9.4 遵循协议.....	560
9.5 功能描述.....	560
9.5.1 随机数生成.....	560
9.5.1.1 NRBG.....	560
9.5.1.2 DRBG.....	560
9.5.2 随机数测试.....	561
9.5.3 随机数性能.....	561
9.6 软件指南.....	561
9.6.1 复位和初始化.....	561
9.6.1.1 复位.....	561
9.6.1.2 硬件初始化.....	562
9.6.1.3 软件初始化.....	562
9.6.2 应用编程接口.....	562
9.6.2.1 测试 AIS-31 Monobit,run,poker 测试电路操作流程.....	562
9.6.2.2 测试 repetition count, Adaptive Propotion 测试电路操作流程.....	563
9.6.2.3 测试 SHA256/SM3 加工功能操作流程.....	563
9.6.2.4 测试 SP 800-90A DRBG-AES/SM4_ECB_256 操作流程.....	565
9.6.2.5 测试标准的 NIST SP 800-90A DRBG 操作流程.....	566
9.6.2.5.1 不带 reseed DRBG KAT 测试流程.....	567
9.6.2.6 带 reseed DRBG KAT 测试流程.....	569
9.6.2.7 获取噪声源数据操作流程.....	571
9.7 性能.....	571
9.7.1 实际测试结果.....	571
9.7.2 吞吐率 (Throughput)	571
9.7.3 延迟 (Latency)	572
9.8 DFX 应用指导.....	572
9.8.1 模块维测.....	572
9.8.1.1 中断上报.....	572
9.8.1.2 消息上报.....	572



9.8.1.3 Debug.....	572
9.9 注意事项.....	573



插图目录

图 1-1 应用框图.....	8
图 1-2 模块框图.....	10
图 1-3 写反压数据结构.....	15
图 1-4 VF 资源映射 a.....	23
图 1-5 VF 资源映射 b.....	24
图 1-6 报文存储结构.....	26
图 1-7 维测信息.....	28
图 2-1 TM 应用图.....	51
图 2-2 TM 模块框图.....	53
图 2-3 TM 读回的 SID 数据结构要求.....	53
图 2-4 TM 队列管理机制.....	55
图 2-5 RED 处理流程图.....	56
图 2-6 RED 丢弃曲线示意图.....	56
图 2-7 TM 端口超时丢弃.....	58
图 2-8 TM 四级队列调度结构示意图.....	59
图 2-9 SP 调度示意图.....	61
图 2-10 DRR 调度示意图.....	62
图 2-11 SP+DRR 调度示意图.....	64
图 2-12 TM 各层级整形方法示意图.....	65
图 2-13 队列流量整形示意图.....	65
图 2-14 保证带宽流量整形示意图.....	66
图 2-15 同级子节点整形类型配置约束.....	67
图 2-16 Trunk 绑定物理端口示意图.....	67
图 2-17 物理端口对应紧急队列示意图.....	68
图 2-18 虚拟逻辑端口对与物理逻辑端口映射关系示意图.....	72
图 2-19 虚拟逻辑端口聚合与物理逻辑端口聚合映射关系示意图.....	72
图 2-20 虚拟 Trunk 端口与物理 Trunk 端口映射关系示意图.....	73
图 2-21 虚拟 TP 端口与物理 TP 端口映射关系示意图.....	73
图 2-22 TM 多模独立复位.....	75
图 3-1 POE 应用示意图.....	99
图 3-2 POE 内部工作示意图.....	101
图 3-3 流拥塞问题.....	113
图 3-4 调度拥塞问题.....	113



图 3-5 CPU 任务预取缓存.....	114
图 3-6 逻辑 CPU 任务调度图.....	119
图 3-7 加速器的任务调度图.....	120
图 3-8 例 1 Group 内优先级配置.....	120
图 3-9 例 2 Group 内优先级配置.....	121
图 3-10 关闭基于集合 MASK 的互斥调度 ATOMIC 流注册示意图.....	122
图 3-11 使能基于集合 MASK 的互斥调度的 ATOMIC 流注册示意图.....	123
图 3-12 SWITCH_NULL 命令执行.....	124
图 3-13 TT 为 ORDER 的 SWITCH_DESCCHEDULE 命令执行.....	125
图 3-14 TT 为 ATOMIC/NULL 时的 SWITCH_DESCCHEDULE 命令执行.....	125
图 3-15 DESCCHEDULE 命令执行.....	126
图 3-16 SEND_TM、SEND_POE 命令执行.....	126
图 3-17 推送 Cacheable 存储空间示意图.....	132
图 4-1 PPE 结构图.....	154
图 4-2 PPE 通道使用场景.....	163
图 4-3 MAC 组网场景.....	163
图 4-4 发送直通 MAC 示意图.....	165
图 4-5 接收直通 CPU 示意图.....	166
图 4-6 MAC 间转发示意图.....	167
图 4-7 CPU 到 CPU 示意图.....	168
图 4-8 MAC 到 CPU 示意图.....	169
图 4-9 CPU 到 MAC 示意图.....	170
图 4-10 MAC 处理的 1588 报文格式.....	172
图 4-11 1588 报文发送单步法.....	174
图 4-12 1588 报文发送两步法.....	174
图 4-13 ROCE 在 PPE 转发处理流程.....	177
图 4-14 封装格式 A.....	178
图 4-15 封装格式 B.....	178
图 4-16 帧格式.....	178
图 4-17 帧格式.....	178
图 4-18 报文转发流程处理.....	182
图 4-19 报文分发流程表.....	183
图 4-20 分发流程图.....	184
图 4-21 自定义分发生成示意图.....	192
图 4-22 优先级映射管理.....	194
图 4-23 自定义 QoS 生成示意图.....	195
图 4-24 QoS 的配置划分.....	195
图 4-25 LSW 转发流程图.....	196
图 4-26 VLAN 识别.....	201
图 4-27 非隧道报文模式.....	201
图 4-28 隧道报文模式.....	201
图 4-29 出口处 VLAN 处理.....	203



图 4-30 流表 key 选择结构图.....	207
图 4-31 查表流程.....	209
图 4-32 Tuple data 域段作为 TCAM 查表数据对齐方式.....	214
图 4-33 Meta Data 域段作为 TCAM 查表方式对齐方式.....	215
图 4-34 Key 产生方式.....	215
图 4-35 CPU 操作流程.....	226
图 4-36 L2 表学习和功能查找.....	227
图 4-37 学习限制 counter 流程.....	229
图 4-38 老化处理流程图.....	230
图 4-39 老化处理流程.....	231
图 4-40 简单异或算法.....	235
图 4-41 Trunk hash 隧道报文 key 选择.....	236
图 4-42 Trunk CRC 运算.....	237
图 4-43 Trunk IP 屏蔽处理.....	237
图 4-44 Trunk TCP/UDP 端口号屏蔽处理.....	238
图 4-45 LSW 入口处理流程.....	241
图 4-46 LSW 查 L2 表流程.....	242
图 4-47 使用 SMAC 查找 L2 表路程.....	243
图 4-48 LSW 转发流程.....	244
图 4-49 LSW 过滤流程.....	245
图 4-50 LSW 出口处理流程.....	246
图 4-51 VLAN 卸载流程.....	247
图 4-52 镜像流程.....	248
图 4-53 按照 QoS 丢弃报文.....	252
图 4-54 接收报文在 DDR 中的存储方式示意图（描述符报文非分离模式）.....	253
图 4-55 接收报文在 DDR 中的存储方式示意图（描述符报文分离模式）.....	253
图 4-56 发送报文在 DDR 中存储方式.....	255
图 4-57 INT header.....	256
图 4-58 IOAM header 的三种格式.....	256
图 4-59 INT/IOAM 业务流程图.....	256
图 4-60 INT over Vxlan-gpe Header（INT header 在 IOAM Option and Data Space）.....	258
图 4-61 IOAM over Vxlan-gpe Header（IOAM header 在 IOAM Option and Data Space）.....	258
图 4-62 INT over Geneve Header（INT header 在 IOAM Option and Data Space）.....	258
图 4-63 IOAM over Geneve Header（IOAM header 在 IOAM Option and Data Space）.....	259
图 4-64 发送方向处理字段.....	259
图 4-65 典型报文的处理流程.....	261
图 4-66 调度结构图.....	282
图 4-67 根据各端口速率配置 SSU 发送侧调度归属 group 相关寄存器原则.....	299
图 4-68 MAG 环回.....	300
图 5-1 SA 加速器连接示意图.....	309
图 5-2 连续字符丢弃示意图.....	312
图 5-3 去重原理示意图.....	313



图 5-4 SA 硬件知识库加载总体流程图.....	332
图 5-5 编译结果单数据段写入流程图.....	333
图 5-6 加载流程.....	334
图 6-1 SA 转换器使用示意图.....	361
图 6-2 SA 转换器处理流程图.....	362
图 6-3 规则转换调用流程框图.....	370
图 6-4 中间结果合并调用流程框图.....	370
图 7-1 TRING 结构框图.....	373
图 7-2 TRING 应用框图.....	376
图 7-3 TRING WHEEL 实现原理.....	377
图 7-4 TRING 上报完成中断图.....	377
图 7-5 step_hdr 最大挂接 TCB 数目分析.....	384
图 7-6 START 任务处理延迟分析.....	384
图 7-7 STOP 任务处理延迟分析.....	385
图 7-8 ADD_POE 消息吞吐量.....	385
图 8-1 SEC 架构简图.....	388
图 8-2 ESP Head.....	399
图 8-3 ESP TAIL.....	399
图 8-4 AH Head.....	400
图 8-5 IPv4 ESP Transfer Mode.....	400
图 8-6 IPv4 ESP 隧道模式.....	401
图 8-7 IPv4 ESP 隧道 Nat 模式.....	402
图 8-8 IPv4 AH 传输模式.....	402
图 8-9 IPv4 AH 隧道模式.....	403
图 8-10 IPsec ESP 传输模式上行方向 SEC 处理.....	403
图 8-11 IPsec ESP 隧道模式上行方向 SEC 处理.....	404
图 8-12 IPsec ESP 传输模式下行方向 SEC 处理.....	404
图 8-13 IPsec ESP 隧道模式下行方向 SEC 处理.....	405
图 8-14 IPsec AH 传输模式上行方向 SEC 处理.....	405
图 8-15 IPsec AH 隧道模式上行方向 SEC 处理.....	406
图 8-16 IPsec AH 传输模式下行方向 SEC 处理.....	406
图 8-17 IPsec AH 隧道模式下行方向 SEC 处理.....	407
图 8-18 OpenSSL/TLS 下行方向 SEC CBC 算法单加密处理.....	407
图 8-19 OpenSSL/TLS 下行方向 SEC CBC 算法单加密处理后插入 IV.....	408
图 8-20 OpenSSL/TLS 上行方向 SEC CBC 单解密处理.....	408
图 8-21 OpenSSL/TLS 上行方向 SEC CBC 单解密处理后去除 IV.....	408
图 8-22 OpenSSL/TLS 上行/下行方向 SEC 认证/认证校验处理.....	409
图 8-23 OpenSSL/TLS 下行方向 SEC 认证+CBC 加密处理.....	410
图 8-24 OpenSSL/TLS 上行方向 SEC CBC 解密+认证校验处理.....	411
图 8-25 OpenSSL/TLS 上行方向 SEC CCM 加密认证.....	411
图 8-26 OpenSSL/TLS 上行方向 SEC CCM 解密解认证.....	412
图 8-27 OpenSSL/TLS 上行方向 SEC GCM 加密认证.....	412



图 8-28 OpenSSL/TLS 上行方向 SEC GCM 加密认证后报文结构.....	412
图 8-29 OpenSSL/TLS 上行方向 SEC GCM 解密解认证.....	413
图 8-30 OpenSSL/TLS 上行方向 SEC GCM 解密解认证后报文结构.....	413
图 8-31 TLS1.3 SEC 加密方向处理.....	414
图 8-32 IPsec AH 隧道模式下行方向 SEC 处理.....	415
图 8-33 补码加法运算规则.....	423
图 8-34 UDP 头结构.....	423
图 8-35 UDP checksum 数据格式.....	425
图 8-36 IPv6 伪头格式.....	425
图 8-37 DDR 中密文 KEY 的数据格式.....	427
图 8-38 一级 UNWRAP 数据格式与操作流程.....	428
图 8-39 SGL 和 SGE 的数据格式.....	469
图 8-40 PPE 数据流示意图.....	470
图 8-41 PPE List 单链示意图.....	471
图 9-1 TRNG 应用框图.....	556
图 9-2 HTRNG IP 的内部逻辑框图.....	557
图 9-3 CTR_DRBG 算法流程.....	561
图 9-4 测试向量.....	564
图 9-5 写入加工函数的 optional additional input.....	564
图 9-6 写入噪声源测试数据.....	564
图 9-7 NIST 测试向量.....	565
图 9-8 写入测试 key 值到 HTRNG_DRBG_PS_AI4~11.....	566
图 9-9 写入测试明文到 HTRNG_DRBG_PS_AI0~3.....	566
图 9-10 CTR_DRBG 测试向量.....	568
图 9-11 第二次写入.....	568
图 9-12 第三次写入.....	568
图 9-13 结果数据.....	568
图 9-14 CTR_DRBG 测试向量.....	570
图 9-15 第二次写入.....	570
图 9-16 第三次写入.....	570
图 9-17 结果数据.....	571



表格目录

表 1-1 Timer 术语表..... 1

表 1-2 Timer 中断产生时序图..... 16

表 1-3 BMU 释放缓存数据结构..... 27

表 1-4 维测信息数据域说明..... 28

表 1-5 内部状态查询..... 40

表 1-6 功能统计..... 40

表 2-1 TM 关键术语描述..... 44

表 2-2 RED 软件配置参数表..... 57

表 2-3 TM 寄存器划分多模列表..... 69

表 2-4 TM 模块相关时钟复位配置寄存器..... 78

表 2-5 软件 PF 初始化需要配置的寄存器列表..... 79

表 2-6 软件 VF 初始化需要配置的寄存器列表..... 81

表 3-1 POE 术语表..... 90

表 3-2 系统中各个模块操作 POE 列表..... 100

表 3-3 处理单元和 Group 映射关系配置举例..... 117

表 3-4 POE 寄存器划分多模列表..... 129

表 3-5 32bit 大小端转变表（ A4/B4/C4/D4 为 MSB ） 137

表 3-6 64bit 大小端转变表（ A8/B8 为 MSB ） 137

表 3-7 POE 失败任务数据结构..... 148

表 4-1 PPE 关键术语..... 152

表 4-2 各速率与频点说明..... 164

表 4-3 8Byte1588 操作符内容..... 172

表 4-4 接收直通表..... 180

表 4-5 发送直通表..... 181

表 4-6 分发流表基于元组产生 key 的待选数据格式..... 185

表 4-7 分发流表的 Meta data..... 188

表 4-8 分发流表的动作..... 193

表 4-9 自定义分发表的数据结构..... 193

表 4-10 单 VLAN 处理处理模式..... 197

表 4-11 Vlan Stacking 模式..... 197

表 4-12 入口 VLAN 和入口流表处理合并..... 199

表 4-13 单 VLAN 模式选择..... 202

表 4-14 默认值处理方式..... 202



表 4-15 出端口 VLAN 处理.....	204
表 4-16 入口 VLAN 处理和出口 VLAN 处理合并规则.....	205
表 4-17 不同位宽流表对应最大深度.....	207
表 4-18 入口流表模板配置.....	208
表 4-19 入口流表 Key 待选数据.....	210
表 4-20 入口流表关联数据.....	215
表 4-21 出口流表 Key 待选数据.....	219
表 4-22 出口流表关联数据.....	222
表 4-23 L2 地址表 Key.....	224
表 4-24 L2 地址表关联数据.....	224
表 4-25 漂移表.....	227
表 4-26 STP 状态表.....	232
表 4-27 RSTP 状态表.....	232
表 4-28 每个 Trunk group 内容.....	233
表 4-29 HASH 选路功能表.....	233
表 4-30 CRC 输入 key.....	236
表 4-31 VLAN 过滤表.....	239
表 4-32 多播复制表.....	239
表 4-33 RSS Key 的选择.....	249
表 4-34 多播 FIFO 配置.....	251
表 4-35 报文长度约束.....	255
表 4-36 INT/IOAM over IPv6 Hop-by-Hop Extension Header.....	257
表 4-37 INT/IOAM over IPv6 Destination Options Extension Header.....	257
表 4-38 GRE、TCP/UDP 报文中封装 INT/IOAM Heade.....	257
表 4-39 TPU 操作.....	259
表 4-40 不同报文类型的 checksum 处理（OL3 for Outer L3，OL4 for Outer L4）.....	261
表 4-41 UDP checksum 为 0 时的处理.....	264
表 4-42 dyinggasp 的帧格式.....	265
表 4-43 PPE ADD_PACKET 数据结构 0x41（Order_TagValue 与 Affinity_TagValue 复用场景）.....	266
表 4-44 ADD_SPE_WORK，PPE 上送 1588 时戳到 NP 的消息数据结构.....	267
表 4-45 发送描述符首片节点数据结构.....	268
表 4-46 发送/接收描述符非首片节点数据结构.....	273
表 4-47 接收描述符首片节点数据结构.....	274
表 4-48 配置寄存器.....	283
表 4-49 Shaping 配置.....	287
表 4-50 网口分布场景.....	299
表 5-1 SA 加速器术语表.....	306
表 5-2 SA BD（cpu 发给加速器的任务描述符）数据结构 64bytes.....	313
表 5-3 buf_info 结构.....	317
表 5-4 匹配结果格式.....	317
表 5-5 ADD_PACKET 数据结构 0x41（带 Affinity，Timestamp，Prefetch 或 Hardware 扩展场景）.....	319
表 5-6 各数据接口大小端模式.....	328



表 5-7 知识库表项分组.....	331
表 5-8 描述信息异常处理机制.....	337
表 5-9 AXI 总线响应 resp 异常中断.....	339
表 5-10 SA 内部 fifio 状态寄存器.....	343
表 5-11 功能统计寄存器.....	347
表 6-1 SA 转换器术语表.....	357
表 6-2 SA 正则语法支持表.....	358
表 6-3 返回码说明.....	371
表 7-1 TRING 术语表.....	374
表 8-1 SEC 加速器术语表.....	389
表 8-2 IPv4 Head.....	397
表 8-3 IPv6 Head.....	398
表 8-4 GCM 算法流模式首中尾 BD 配置流程.....	416
表 8-5 CTR 算法流模式首中尾 BD 配置流程.....	417
表 8-6 报文在 DDR 中大端存储方式.....	420
表 8-7 IPSEC、OpenSSL/TLS 场景下 IVIN 组成方式.....	422
表 8-8 IPv6 Header 结构.....	424
表 8-9 SQE 数据结构（除存储重删及上下盘加解密以外的场景，如下为小端格式描述）.....	429
表 8-10 SEC 输入包任务描述符.....	462
表 8-11 SEC 输出包任务描述符.....	464
表 8-12 BD FIFO 任务控制组合.....	475
表 8-13 AES-CCM 算法参数配置.....	480
表 8-14 AES-CCM 算法参数配置.....	480
表 8-15 AES-GCM 算法参数配置表.....	481
表 8-16 SEC 常见异常处理.....	489
表 8-17 未检测异常 BD 处理.....	553
表 9-1 TRNG 关键术语描述.....	557
表 9-2 DRBG 随机数产生吞吐率.....	571
表 9-3 DRBG 随机数产生延迟.....	572



1 BMU

1.1 概述

包缓存管理单元（BMU）是一个自动化的缓存管理系统，主要在简化缓存申请与释放的过程，从而降低软件开销。

- 1. BMU支持动态注入内存块或Buffer，BMU可以自动分割Buffer以管理内存单元，或直接接受管理已注入的Buffer。
- 2. 在接收数据包时，硬件加速器能够向BMU请求所需的Buffer空间，并且可以通过指定的Pool来获取Buffer空间，BMU将会成功申请到的Buffer指针反馈给硬件加速器。CPU在需要Buffer时，也可向BMU发起申请Buffer空间。
- 3. 当Buffer不再被使用并需要释放时，无论是硬件加速器还是CPU，都可以将需要释放的Buffer指针送回BMU，由BMU负责执行实际的Buffer释放操作。

1.2 关键术语

表1-1为与BMU相关的关键术语表。

表 1-1 Timer 术语表

术语	描述
Pool	资源池，缓存空闲Buffer空间的指针，指针位宽48bit。
Buffer	内存的存储空间块，用来存储包内容。
LIFO	Last In First Out（后进先出，一种类似于堆栈行为的缓存结构）。
VFID	VF编号信息，在Hi309a主要用于多模的模式（例如GSM、UMTS、LTE-FDD、LTE-TDD）区分。
AXID	AXI总线ID。
LPID（维测信息中）	此处的LPID不是逻辑Pool ID，指的在维测信息中用于区分一个Cluster内的CPU核号或者用于区分一个subsys上的加速器编号。



术语	描述
SRCID	用于区分一个Cluster号或者用于区分一个subsys的编号。
Block	物理Pool共享内存管理的最小单位，每个Block对应连续的4MByte或8MByte的内存空间。
QoS	quality of service，服务质量。
Tcam	Ternary Content Addressable Memory，三重内容可编址存储器。
Mbist	Memory build in self test。
DFT	Design for test。
物理Pool	BMU共享内存管理的基本载体，动态管理软件注入的Block，每个物理Pool可以对应一个或多个逻辑Pool。
逻辑Pool	逻辑Pool隶属于物理Pool，逻辑Pool共享物理Pool所管理的Block，BMU共32个逻辑Pool。
虚拟物理Pool	物理Pool由硬件按照映射表转换后的物理Pool。
虚拟逻辑Pool	逻辑Pool由硬件按照映射表转换后的逻辑Pool。
半静态配置	只允许在模式初始化时修改配置。
静态配置	只允许在设备初始化时刻修改配置。
动态配置	允许任意时刻修改配置。
DFS	Dynamic Frequency Scaling。
TZPC	TrustZone Protection Controller。

1.3 特性

BMU模块具有以下特点：

- 支持16个物理Pool，32个逻辑Pool。
 - 每个物理Pool可以关联一个或多个逻辑Pool，但每个物理Pool最多只能关联32个逻辑Pool。
 - 每个逻辑Pool只能归属于一个物理Pool，或者由软件进行注入或获取。
 - 逻辑Pool与物理Pool之间的绑定关系可以通过软件进行半静态配置。
 - 系统支持一个VF（虚拟功能）对应一个或多个物理Pool，单每个物理Pool只能归属于一个VF。
 - 在Hi309a设备中，仅使用逻辑Pool0~15，PPE（Packet Processing Engine）侧不会处理逻辑Pool16~31的QoS反压信号。
- 每个物理Pool的最大内存支持取决于系统配置和内存管理模式。
 - 在8Mbyte Block内存管理模式：
 - 若所有逻辑Pool的Buffer Size均大于等于2kByte，则每个物理Pool最大支持2GByte空间，且所有物理Pool均最大支持2GByte空间。



- 若所有逻辑Pool的Buffer Size均小于2kByte, 则每个物理Pool最大支持256MByte空间, 且所有物理Pool最大支持256MByte空间。
 - 在4MByte Block内存管理模式下:
 - 若所有逻辑Pool的Buffer Size均大于等于1kByte, 则每个物理Pool最大支持1GByte空间, 且所有物理Pool最大支持1GByte空间。
 - 若所有逻辑Pool的Buffer Size均小于1kByte, 则每个物理Pool最大支持256MByte空间, 且所有物理Pool最大支持256MByte空间。
 - 每个物理Pool支持最多256个不连续的内存地址区间配置, 即256个Block。
 - 物理Pool管理的空间大小可动态配置, 支持动态注入和删除Block。
 - 支持多个逻辑Pool可以共享一个物理Pool的空间。
 - 支持48位内存地址, 最大支持管理任意256个Block空间的地址覆盖, 管理的Buffer地址可以是VA、IPA或PA, 系统需要保证SMMU的地址映射正确。
- 在逻辑Pool的内存管理和Buffer管理中, 存在硬件管理模式和软件管理模式这两种模式的具体特性。
 - 硬件管理模式:
 - 每个逻辑Pool所管理的内存空间不能超过其所在的物理Pool管理的内存空间, 同时, 一个物理Pool内所有逻辑Pool的内存总和也不能超过该物理Pool的内存容量。
 - 单一逻辑Pool能够处理多达1M个Buffer, 并且整个系统中所有逻辑Pool的Buffer总数同样不超过1M个。
 - 硬件管理模式允许对逻辑Pool占用物理Pool空间的最大值和最小值进行动态调整。
 - 逻辑Pool可以根据物理Pool的Block的资源及预设的申请和释放Block阈值, 动态地向物理Pool请求和归还Block。当逻辑Pool内的空闲Buffer数量低于设定的申请阈值时, 将向物理Pool请求Block; 反之, 当空闲的Buffer数量超过释放阈值并且有完整的Block可用时, 会释放Block给物理Pool。这些阈值可以随时更改。
 - 在Buffer申请和释放流量相对稳定的情况下, 硬件管理模式支持基于Block局部性的Buffer申请策略, 以确保逻辑Pool优先使用最近频繁访问的Block来获取Buffer。
 - 软件管理模式:
 - 软件管理模式, 每个逻辑Pool的内存容量取决于软件注入的Buffer数量及其大小。
 - 在这种模式下, 每个逻辑Pool最多能支持1M个Buffer, 但具体数目受限于LIFO/FIFO空间以及溢出回读空间, 对于FIFO存储模式, 不支持溢出回读机制, 且最多只能容纳4096个Buffer; 而LIFO存储模式则支持溢出回读, 并允许设置溢出回读阈值。
 - 软件管理模式, 逻辑Pool无需与特定的物理Pool绑定, 而由软件维护对应的Buffer, 此外, 逻辑Pool的空间可被配置为FIFO/LIFO行为, 并在注入过程中支持地址边界检查。
 - 物理Pool的共享内存管理以Block为最小基本单位。



- 系统支持4MB Block和8MB Block两种内存管理模式，这两种模式之间是互斥的，一旦选定一种模式，所有物理Pool都将遵循这一设定，该配置属于芯片级设置，软件只能进行静态配置模式。
- 若系统采用支持4M Block模式，则每个Block将占用连续的4MB内存，并且Block的起始地址必须是4MB的倍数；如果采用8MB Block模式，则每个Block将占用连续的8MB内存，其起始地址也需是8MB的倍数。
- 硬件会自动在逻辑Pool内对Block进行Buffer分割，当逻辑Pool在分割Block时出现无法整除的情况，将会导致小于逻辑Pool的Buffer 大小的内存被浪费掉。
- 在硬件管理模式下，每个逻辑Pool的缓存长度可以静态配置，并且各个逻辑Pool之间可以独立设置。配置值需为256Byte的整数倍，最大支持长度为64kB，支持的Buffer长度范围是256Byte~64kB（以256Byte为增量单位）。而在软件管理模式下，则支持对大于64Byte（以64Byte为增量单位）的任意长度Buffer进行管理。
 - 当加速器或CPU申请缓存时，会根据所需长度向相应的逻辑Pool提出请求。
- 支持逻辑Pool缓存分配的后进先出特性（LIFO，也称为堆栈特性）。
- 支持缓存请求者指定逻辑Pool申请Buffer指针。
- 支持逻辑Pool的Buffer释放，Buffer指针所属的逻辑Pool由硬件通过命令接口映射VFID，并通过查询内部表项确定，具体接口请参阅释放Buffer章节描述。
- 支持AXI Slave接口。
 - 支持读/写寄存器操作：采用单个突发（Burst）模式，每次传输32位数据。
 - 向逻辑Pool申请Buffer时，可一次申请1个、2个或4个Buffer。具体操作包括：64位突发模式、128位突发模式和256位突发模式。
 - 向逻辑Pool释放Buffer时，支持一次释放单个或者多个连续的Buffer（链式释放）。64位突发模式用于释放单个缓冲区，以及256位突发模式用于一次性释放四个单个缓冲区。
 - 向物理Pool添加Block时，使用64位突发模式进行操作。
 - 向物理Pool删除Block时，同样采用64位突发模式进行操作。
 - AXI接口支持48位宽的总线地址。
 - AXI slave共支持16深度的缓存FIFO，供CPU和加速器共同使用。
- 支持多种Endian模式：
 - 寄存器访问仅支持小端模式。
 - CPU的申请、释放接口以及注入Block接口支持大小端模式配置。其中，大端模式仅限于32位和64位大端模式，且默认采用32位大端模式。
 - 加速器访问仅支持小端模式。
 - 写入诊断信息及读娶解析描述符的操作在访问DDR空间时，其大小端模式可配置，并需与全局数据结构中的解析描述符保持一致。此外，每个虚拟功能（VF）可独立配置大小端模式，大端模式同样支持32位大端和64位大端模式，默认情况下使用32位大端。
 - 在执行链式释放过程中对DDR的溢出或回读操作时，也统一采用小端模式。
 - 软件管理模式下对DDR的溢出或回读操作时，也统一采用小端模式。
- 支持多模隔离和虚拟化特性。
 - 支持操作系统将物理Pool按VF进行分组。同一个物理Pool中的逻辑Pool必须属于相同的VF，物理Pool与VFID之间绑定关系仅支持半静态配置。
 - 提供了针对加速器的VF反压链式Buffer和分离式Buffer释放的支持。



- 支持逻辑Pool缓存回收及重新初始化。软件通过按VF业务复位，可以完成该VF下所有逻辑Pool和物理Pool复位。这实现了单一模式下的复位功能，复位后物理Pool和逻辑Pool将被清空，需要由软件重新进行初始化。
- 支持各个模式的中断独立报告，并且中断信息按模式存储。
- 在上报中断时，提供异常逻辑Pool ID，由于逻辑Pool ID与VFID相关联，可以通过映射关系确认对应的虚拟Pool ID，并将中断上报给相应的VF。
 - 对于因基于逻辑Pool操作触发的异常中断类型，在中断信息中提供异常的逻辑Pool ID，例如软件尝试向同一个逻辑Pool重复释放相同Buffer的情况。
 - 对于无法确认逻辑Pool的操作触发的异常中断类型，在中断信息中不提供异常的逻辑Pool ID，例如软件释放了一个不属于任何逻辑Pool的Buffer的情况。
- 支持以下异常中断的上报。
 - 内存相关中断：包括LIFO RAM配置冲突中断、内部RAM ECC校验错误中断、Buffer映射除不尽中断。
 - 逻辑Pool相关中断：包括逻辑Pool中断、逻辑Pool内部空中断、访问未启动逻辑Pool中断。
 - 队列管理相关中断：包括QoS水线全反压中断、重复释放中断、边界中断、假满反压中断。
 - Buffer管理相关中断：链式Buffer超长链中断，读数据污染中断、非法读访问中断、非法写访问中断、魔术字校验失败中断、软件管理模式真满、软件管理模式假满、软件管理模式注入/释放Buffer真满丢弃中断。
 - 其他中断：包括状态机超时中断、总线错误中断、链式Buffer释放PPE发送方向描述符OWN域非法中断。
- 支持针对32个逻辑Pool的每一个，分别为其内的9个QoS参数独立配置双门限丢弃策略，且每条策略的最小调整单位为64个Buffer。具体功能包括：
 - 能够生成成用于指示是否触发丢弃操作的信号。
 - 根据申请者所设定的QoS参数来确定相应的门限值。
 - 向PPE提供一个指示，表明每个逻辑Pool内所有QoS参数均已达到反压阈值的状态，并允许通过寄存器访问该状态信息。
 - 在Hi309a设备中，PPE端仅处理逻辑Pool编号从0~15之间的QoS反压通知。
- 硬件管理模式支持缓存重复释放监控，而软件管理模式不支持重复释放检测。
- 提供系统缓存Bitmap和LIFO/FIFO中的Tcam的监控功能，支持对包的入口缓存和出口缓存进行实时监控。
 - 0表示该bit对应的Buffer已分配。
 - 1表示该bit对应的Buffer空闲，待分配。
- 缓存管理机制如下：
 - 在分配包缓存时，检查Buffer对应的位是否为0b1，并将其置为0b0；在释放包缓存时，检查Buffer对应的位是否为0b0，并将其置为0b1。
 - 在Bitmap的重复释放监控期间，如果出现检测位不符，将上报中断。
 - 对于重复释放的Buffer，不执行释放操作，直接丢弃。
 - 最多支持1M个Buffer指针监控。
- 支持生成可用于检查缓存泄漏或内存踩踏的维测信息。



- 维护检测信息应包含以下字段：SRC_ID、时间戳、LPID、虚拟逻辑Pool ID、魔数（MAGIC_NUM）、5bit的申请、释放操作携带的自定义字段。
- 系统需具备将维测信息写入外部动态随机存取存储器（DDR）的能力。
- 在执行申请和释放操作时，硬件能够自动更新维测信息。同时，软件可根据逻辑Pool的具体配置决定是否在这些操作中更新维测信息。
- 当系统处于硬件管理模式时，BMU应当支持使用魔数。在Block时，需要填充魔数；当Block归还时，应对魔数进行校验，以确保Block的所有Buffer未遭受覆盖。若发现有覆盖现象，则应触发中断，并记录首次被覆盖的Buffer地址及其所属Block的起始地址。只有在完成所有Buffer的验证后，才能将Block归还给物理Pool。对于一个典型的含有4096个缓冲区的块，魔数的插入和验证过程大约需要50微秒的时间。魔数的存在会影响Block的申请、释放、VF复位和Buffer碎片化回收的操作延时，具体延时计算公式为：（所需处理的Block数+前级待处理的Block数）*50微秒，对于不同大小的块，其延迟时间则根据Buffer Size的Block延时为对应（Buffer个数/4K）*50微秒。
- 时间戳采用32位宽度，提供1s的计数精度，最大计时可达4G秒。
- 由于总线阻塞可能造成维测信息丢失，因此系统应允许软件配置是否启用AXI Slave设备的反压机制，使能反压后，维测信息就不会丢失。
- 该系统支持链式Buffer和分离式Buffer的释放操作，具体特性如下：
 - 支持单Buffer、链式Buffer和分离式Buffer的释放。
 - 提供VF0~4共5个制式的业务队列。
 - 允许独立配置各VF的描述符大小端解析方式。
 - 支持多模式复位，可根据VF队列独立执行复位操作。
 - 软件可配置链式Buffer的DDR溢出回读基地址和缓存空间深度，按VF队列独立配置。
 - 支持VF共5个业务队列共享片内缓存，总共可缓存1024个需要释放的链式Buffer首指针或分离式Buffer首指针，每个队列的溢出回读水线和反压水线可独立配置。
 - 按VF对需要链式释放的Buffer头进行计数。
 - 提供链式Buffer释放的极限带宽在每秒15M个链式节点Buffer（假设DDR访问平均延时在150个CPU cycle）。
 - 加速器可能会返回假满状态。
- 支持统一的输入输出（I/O）及硬件加速器计算（HAC）加速器访问接口。
 - BMU的中断地址和中断编号与所有加速器保持一致。
 - 提供软件查询包硬件规格信息的寄存器支持。
 - 支持软件对设备号PF_ID和功能号VF_ID的读写操作。
 - 采用统一的虚拟化VF接口标准。
 - 统计类寄存器提供写清除功能。
- 在Hi309a设备上启用写维测模式后，缓存申请与释放的带宽均为15Mpps，能够满足CPU和HAC的并发操作需求。
- 所有TCAM模块均具备Mbist测试能力，在DFT模式和function模式下均可进行Mbist扫描（硬件逻辑支持）。
- 该设备还集成了DFX特性。
 - 支持物理Pool内部状态查询，包括空闲Block数量及详细信息。
 - 提供逻辑Pool内部状态查询，涵盖Block总数及实时Buffer数量等统计数据。



- Bitmap具备自我修复能力，能够识别并丢弃重复释放的Buffer，并记录相关错误地址和其他相关信息。
- 提供VF注入及删除操作中的Block数量统计。
- 实施地址合法性验证机制，根据预设地址范围检查释放Buffer地址的有效性。
- 在软件管理模式下，支持Pool内Buffer地址边界的检查。
- 硬件管理模式下，释放Buffer时执行与TCAM绑定的Buffer重复释放检查。
- 总线访问性能监测，包括：
 - 单词读/写操作的最大延迟时间。
 - 读/写操作的平均延迟时间。
 - 读/写操作的outstanding请求计数。
 - 读/写操作的总次数统计。
- 内部状态机可视化，便于诊断模块是否出现挂死的情况。
- 状态机具有智能心跳功能，若某一个状态机在非空闲状态下停留超过设定阈值，则触发心跳停止信号，并允许状态查询。
- 内部SRAM数据可由CPU间接读取，不影响原有存储与控制状态，但禁止直接写入操作。
- 设备提供全局使能和关闭控制，未启用状态下无法进行初始化操作，从而影响其正常运行。
- 支持单独开启或关闭逻辑Pool。
- BMU复位过程中，可配置申请Buffer操作的响应结果。
- 所有内部SRAM表项均配备自检功能，支持1bit ECC自动纠错、同时区分报告1bit 和2bit ECC错误，静态表项（包含RAM和TCAM）的1bit ECC错误可自行修正。
- 异常中断上报机制；记录首次发生错误的状态信息。
- 能够检测并报告重复释放Buffer的故障警告。
- 监控发送至总线的命令，并在mst_ooo寄存器中记录。
- 提供Debug Bus接口，用于检测状态寄存器及状态机状态等。
- 支持单制式复位模式，强制回收所有逻辑Pool缓存并重新初始化。
- 支持低功耗特性，包括模块级时钟的自动动态关闭和开启。

1.4 遵循协议

遵循AMBA4.0 协议。

1.5 功能描述

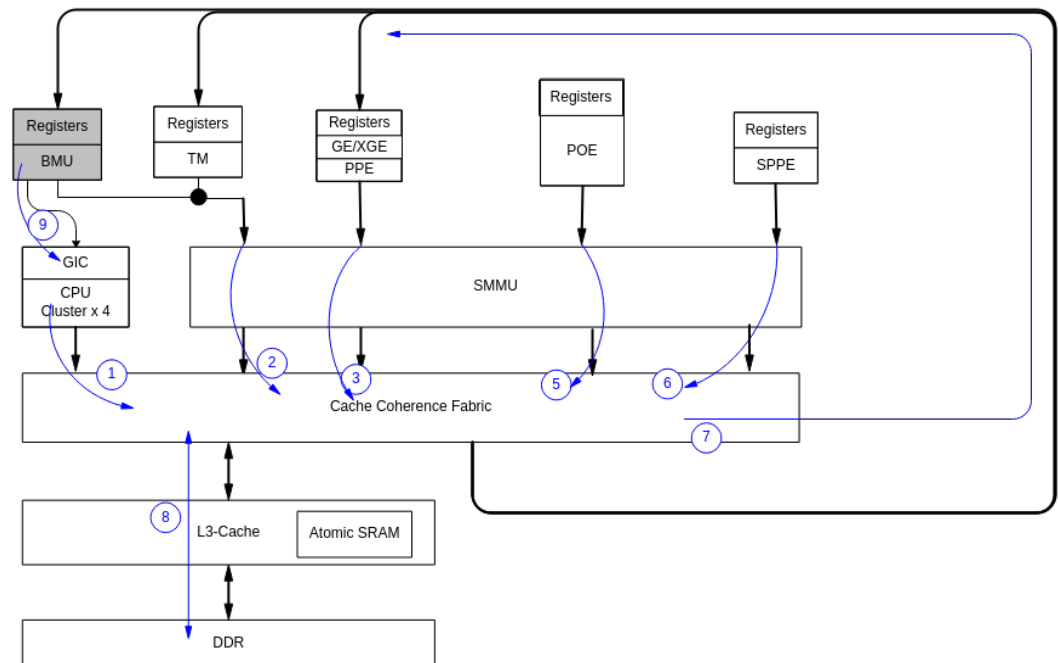
无。

1.5.1 典型应用

1.5.1.1 应用框图

1. CPU Core通过①和⑦线路配置BMU、从BMU申请Buffer或释放Buffer。
2. SPPE通过⑥和⑦线路向BMU申请Buffer或释放Buffer。
3. PPE通过③和⑦线路向BMU申请Buffer或释放Buffer。
4. TM通过②和⑦线路向BMU释放Buffer。
5. POE通过⑤和⑦线路向BMU释放Buffer。
6. BMU通过②和⑧线路对DDR进行读写（维测信息需要写DDR SDRAM，链式Buffer解析需要读DDR SDRAM，待释放链式Buffer队列的溢出/回读需要读写DDR SDRAM）。
7. BMU通过⑨线路上报中断。

图 1-1 应用框图



例如，在PPE接收包时，从逻辑Pool申请Buffer，将包缓存在Buffer中。

在PPE发送包时，当包被处理后，PPE将包通过接口发送出去后，PPE将Buffer的指针释放回BMU（Hi309a 不要求释放Buffer时指定所属的逻辑Pool编号）。

在包的处理过程中，CPU Core可以根据需要向逻辑Pool申请Buffer使用，使用后释放回BMU，或让PPE发送包后将对应Buffer释放回BMU。

在包的处理过程中，包可能会被CPU Core或TM、PPE、SPPE等模块丢弃，包丢弃后，CPU Core或丢弃包的模块需将丢弃包的Buffer释放回BMU。

相关模块

目前在系统中，BMU模块会和下面多个模块进行交互，分别描述如下：

- PPE：PPE通过系统总线向BMU申请Buffer或释放Buffer。

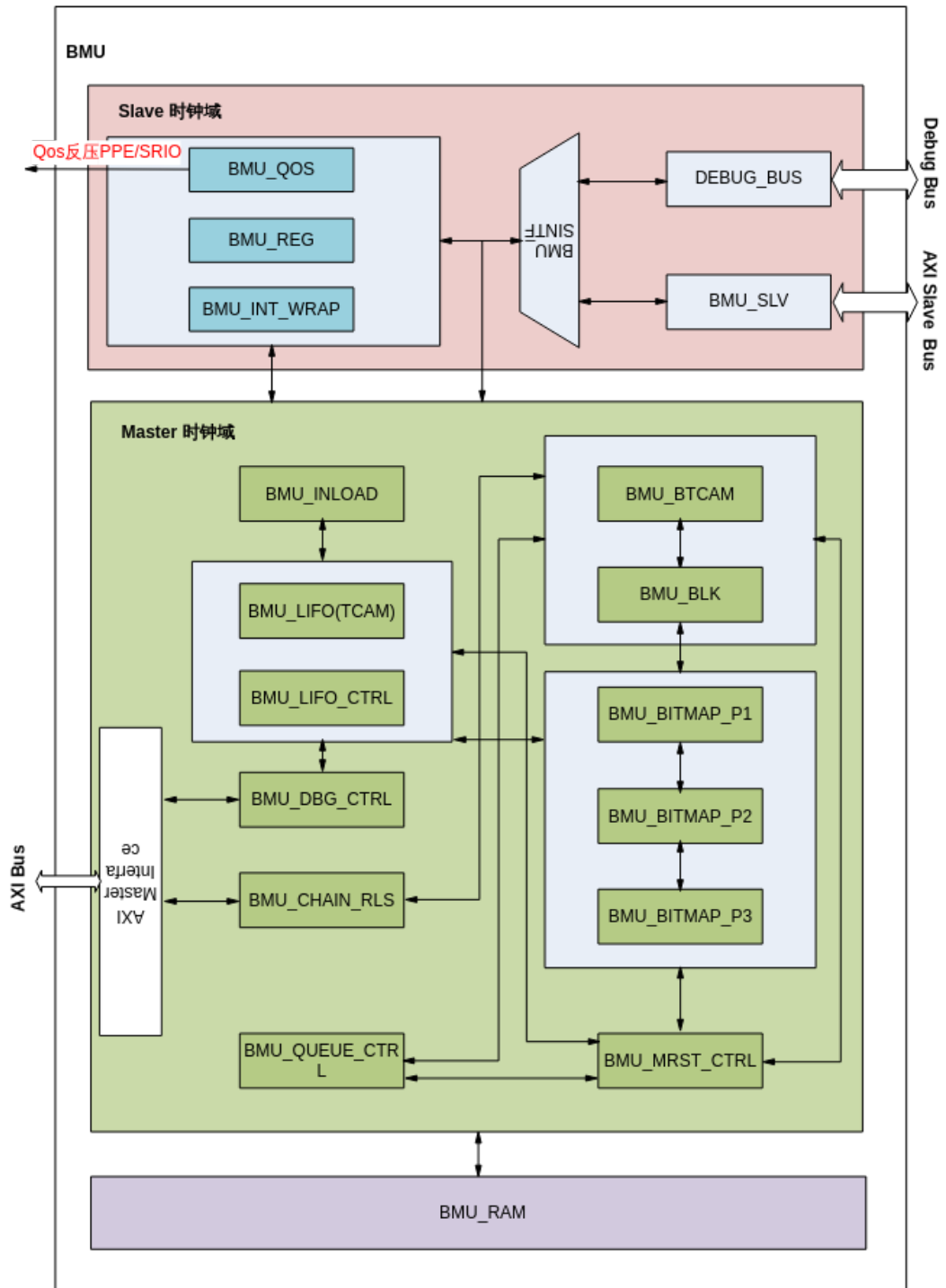


- CPU Core: CPU Core通过系统总线向BMU申请Buffer或释放Buffer。
- TM: TM内部丢弃报文，通过系统总线向BMU释放Buffer。
- POE: POE通过系统总线向BMU释放Buffer。
- DDR: BMU通过系统总线向DDR更新维测信息，链式Buffer和分离式Buffer解析需要读DDR，分离式Buffer回写DDR，待释放链式Buffer的队列溢出/回读需要读写DDR，软件管理模式溢出回读DDR。
- SPPE: 内部生成或者终结报文时，通过总线向BMU申请或释放Buffer。

1.5.1.2 模块框图

32bit/64bit Timer作为硬件定时器，可用于主从串口通信，波特率为187500bps。

图 1-2 模块框图



1.5.1.3 申请 Buffer

当加速器或者CPU需要向BMU申请Buffer时，该模块通过读对应逻辑Pool的寄存器地址来向指定的逻辑Pool申请Buffer，BMU根据地址解析出对应的逻辑Pool编号，从对应逻辑Pool的LIFO中读出Buffer指针反馈给申请模块。BMU对于申请缓存的处理机制：



通过AXI总线的SIZE和LEN区分一次操作申请几个Buffer，BMU支持一次申请操作给出1个、2个或4个Buffer。

向逻辑Pool申请Buffer支持：

- burst1 64bit操作burst1 128bit操作burst1 256bit操作。
- 申请成功：返回Buffer指针。
- 逻辑Pool没有缓存：资源耗尽，给加速器和CPU返回数据全0。
- 逻辑Pool的LIFO空：内部LIFO空，待回读补充Buffer，返回给CPU可配置的值（BMU_EMPTY_DATA），返回给加速器全0。
- VF复位或者BMU独立复位时，返回可配置值（BMU_RST_BP_DATA）给CPU，返回全0给加速器。
- 加速器或CPU申请Buffer时，BMU只返回OKAY响应。

建议发起申请缓存的加速器处理机制：

- 收到返回数据为0后认为申请失败，一次申请多个Buffer，会遇到返回部分Buffer为0的情况，仅需丢弃为0的Buffer，不为0的Buffer可正常使用，禁止丢弃，否则会造成Buffer丢失。暂停可配置时间后再申请，连续失败次数达到可配置次数后上报中断。

加速器与CPU访问的逻辑Pool地址区别如下：

须知

BMU支持5个有效的VF，每个均有独立的申请释放Buffer接口和注入删除Block接口，各VF的对应接口详见多模及虚拟化章节。

加速器申请Buffer时，逻辑Pool对应的BMU的接口寄存器为BMU_RADDR_P，该寄存器接口访问地址为：

{bmu_base_addr[47:0]} + {0x0, 0x14, 逻辑Pool ID[4:0], 0x0}。

对应配置给加速器的基地址为：bmu_base_addr+0x5000。

CPU申请Buffer时，逻辑Pool 对应的BMU的接口寄存器为BMU_CPU_P，该寄存器接口访问地址为：

{bmu_base_addr[47:0]} + {0x0, 0b1, UDF[4:0], 逻辑Pool ID[4: 0], 0x0}。

其中，bmu_base_addr是系统配置给加速器或CPU申请BMU Buffer的基地址，逻辑Pool ID[4:0]的取值范围为0~31，表示32个逻辑Pool（Hi309a仅支持16个逻辑Pool，仅使用低16个地址0~15）；UDF[4:0]是软件的自定义域段，BMU将会将这一域段的信息作为维测信息的一部分写到片外的DDR。

须知

每个逻辑Pool统一配置寄存器BMU_POOL_INIT_RDY[bmu_buf_64b_rev_en]，指示是否有维测信息管理头（64Byte），如果配置为有维测头，申请、释放的接口中的地址是维测信息头之后部分对应的Buffer指针。



1.5.1.4 释放 Buffer

CPU或加速器向BMU释放Buffer时，通过写寄存器释放Buffer，BMU将指针写入对应逻辑Pool的片内LIFO队列中。BMU对于释放缓存的处理机制：

通过AXI总线的SIZE和LEN区分一次操作释放几个Buffer，通过REL域区分单Buffer或非单Buffer释放，BMU支持一次释放1个单Buffer或链式Buffer，加速器接口同时支持一次释放4个单Buffer。

向逻辑Pool释放单Buffer支持：

- burst1 64bit操作burst1 256bit操作。
- 一次释放4个Buffer的操作仅支持加速器接口，且仅供PPE使用。
- 一次释放4个Buffer的操作仅支持单Buffer，对数据结构中REL不为0的Buffer进行丢弃。

加速器释放Buffer时，对应BMU的寄存器BMU_ACC_WADDR，BMU通过Rel域来区分单Buffer、非单Buffer，该寄存器接口访问地址为：

{bmu_base_addr[47: 0]} + {0x00000000, 0x00000028, 5h1E, 0x0};

对应配置给加速器的基址为：bmu_base_addr+0x51E0;

CPU释放Buffer时，对应BMU的寄存器BMU_CPU_WADDR，BMU通过Rel域来区分单Buffer、非单Buffer，该寄存器接口访问地址为：

{bmu_base_addr[47: 0]} + {0x00000000, 0b1, UDF[4: 0], 6h1E, 0x0};

加速器向BMU释放缓存：

- 释放单Buffer：BMU通过总线返回OKAY（0b00）响应。
- 释放非单Buffer：如果非单Buffer释放成功，BMU通过总线反馈OKAY响应，否则返回假满（0bb11）响应，加速器需要重传返回假满响应的Buffer。

CPU向BMU释放缓存：

- 释放单Buffer：BMU通过总线返回OKAY（0b00）响应。
- 释放非单Buffer：BMU通过总线返回OKAY（0b00）响应，若有假满反压，则待反压撤离后返回OKAY响应。

在BMU中提供16个任务缓存，用来吸收总线上来自CPU的outstanding写操作和加速器的outstanding写操作，将添加任务全部接收，但暂时不返回总线response，完成地址译码后返回response。CPU和加速器添加的任务（包含单Buffer、链式Buffer）响应处理是一起的，会相互阻塞。

当逻辑Pool片内LIFO队列达到溢出水线时（溢出水线由寄存器BMU_WL_P配置），根据寄存器BMU_OVRB_BUFFER_NUM_P配置的数量将片内LIFO队列中的Buffer溢出。

BMU对于释放缓存错误的处理机制：

- 除不尽的Buffer指针：丢弃Buffer，上报中断，记录最早的一个任务，Buffer 指针+LPID+SRCID+逻辑Pool ID。
- 指针范围非法：释放缓存的地址与所有逻辑Pool 的Block地址相比较。如果都不命中，认为是释放访问非法，丢弃Buffer，上报中断，记录最早的一个任务，Buffer 指针+LPID+SRCID。
- 重复释放：丢弃Buffer，上报中断，记录最早的一个任务，Buffer指针+LPID+SRCID+逻辑Pool ID。



- 向未使能逻辑Pool释放Buffer: 丢弃Buffer, 上报中断, 记录最早的一个任务, Buffer指针+LPID+SRCID+逻辑Pool ID。

须知

- BMU做重复释放检查时可能由于Bitmap和LIFO之间的Buffer转移的时间差, 以及预取FIFO不做重复释放检查, 无法准确判断重复释放, 但该问题发生为小概率事件。
- 由于释放Buffer时BMU提前返回响应给CPU/加速器, 存在逻辑Pool的资源计数器的更新时延(待Buffer真正回收后才更新计数器), 因此在测试时软件需要增加延时函数去判断统计计数器的值, 推荐时延10ms。

1.5.1.4.1 释放链式 Buffer

Hi309a BMU支持接收链式报文的首Buffer, 并通过解析描述符将链上所有Buffer逐个释放。

释放的必须是首Buffer, 不支持逆向的链表遍历。若不是链式报文的首Buffer, 将可能导致缓存丢失; 支持对循环链表的异常检测, 当一条链的解析节点数超过软件配置值(超长链为循环链的可能性很大)时上报中断, 终止当前链的解析, 并记录错误VFID+LPID+SRCID+Buffer指针。

支持对无法及时处理的链表首Buffer溢出到DDR, 在能够处理时再从DDR回读。

加速器或者CPU通过不同的寄存器接口访问BMU, 释放链式报文的首Buffer(请参见“[释放Buffer](#)”)。

判断为链式报文的首Buffer时, 依次检索链上的各Buffer对应的解析描述符中的NextDataAddr[47:0], 并依次释放链上的全部RefItselfAddr[47:0], 直到解析描述符中读取NextDataAddr[47:0]的值全为0, 完成该链的检索。要求1个链式报文的所有Buffer都属于同一个VF。

1.5.1.4.2 释放分离式 Buffer

Hi309a BMU支持接收分离式报文的首Buffer, 支持通过解析描述符将链上所有BMU管理的Buffer逐个释放回逻辑Pool缓存池, 同时支持解析描述符完毕后回写DDR, 用此方法通知CPU回收CPU管理的Buffer。

释放的必须是首Buffer, 不支持逆向的链表遍历。若不是分离式报文的首Buffer, 将可能导致BMU和CPU管理的缓存丢失; 支持对循环链表的异常检测, 当一条链的解析节点数超过软件配置值(超长链为循环链的可能性很大)时上报中断, 终止当前链的解析, 并记录错误VFID+LPID+SRCID+Buffer指针。

支持对无法及时处理的分离式首Buffer溢出到DDR, 在能够处理时再从DDR回读。

加速器或CPU通过不同的寄存器接口访问BMU, 释放链式报文的首Buffer(请参见“[释放Buffer](#)”)。

判断为分离式报文的首Buffer时, 依次检索链上的各Buffer对应的解析描述符中的NextDataAddr[47:0], 直到解析描述符中读取NextDataAddr[47:0]的值全为0, 完成该链的检索。



1.5.1.5 物理 Pool 动态注入 Block

CPU向物理Pool注入Block时，通过写寄存器方式注入Block的内存基地址，BMU将Block的内存基地址添加到对应物理Pool的队列中。

CPU向物理Pool注入Block时，对应BMU的寄存器BMU_BLK_WADDR_P，该寄存器接口访问地址为：

{bmu_base_addr[47:0]} + {32' h0, 7' h38, 物理Pool ID[4:0], 4' h0}。

CPU向BMU的物理Pool注入Block：

- BMU通过总线返回OKAY (0b00) 状态。
- 软件需要查询寄存器BMU_BLK_INJECT_STATUS获取注入状态。

须知

- VF复位期间，不允许动态注入正在复位的VF的Block，否则现象不可预知。
- 独立复位期间，不允许动态注入任何VF的Block，否则现象不可预知。

1.5.1.6 物理 Pool 动态删除 Block

CPU动态删除物理Pool的Block时，通过读寄存器方式删除物理Pool的Block，BMU将Block从物理Pool的队列中删除。

CPU删除物理Pool的Block时，对应的BMU的寄存器BMU_BLK_RADDR_P，该寄存器接口访问地址为：

{bmu_base_addr[47:0]} + {32' h0, 7' h30, 物理Pool ID[4:0], 4' h0}。

CPU删除物理Pool的Block：

- BMU通过总线返回OKAY (0b00) 状态。
- 软件查询寄存器BMU_BLK_DELETE_STATUS获取删除Block状态。

须知

- VF复位期间，不允许动态删除正在复位的VF的Block，否则现象不可预知。
- 独立复位期间，不允许动态删除任何VF的Block，否则现象不可预知。

1.5.1.7 软件管理模式

BMU支持软件管理模式：软件管理模式逻辑Pool被软件注入/删除Buffer；

软件管理模式，逻辑Pool不与物理Pool绑定，由软件维护对应逻辑Pool的Buffer，并支持注入时地址边界检查；

软件管理模式，可配置为FIFO/LIFO行为，FIFO行为单个逻辑Pool最大支持2048个Buffer指针的FIFO空间，所有的逻辑Pool最大支持4096个Buffer指针的FIFO/LIFO空间；软件管理模式LIFO空间达到溢出水线时，溢出Buffer到软件配置的DDR空间，



LIFO空间低于回读水线时，从软件配置的DDR空间回读Buffer，每个逻辑Pool LIFO溢出回读空间独立可配，以64Byte为粒度，一个64ByteDDR空间可溢出8个Buffer；

软件管理模式BMU会根据配置的对应Pool的高低地址判断释放的Buffer属于哪个逻辑Pool，要求软件管理模式配置给BMU的地址空间相互不踩踏，且不与硬件管理模式下的Block空间踩踏；

软件管理模式，动态注入Buffer时，若开启BMU写维测，需在该Buffer前要给BMU提前预留一个cacheline的维测信息空间，假设注入Buffer的地址为0x0040，BMU将写维测写入0x0000的地址；

软件管理模式不支持魔术字相关功能，需要软件自己填写和校验；

软件管理模式LIFO存储模式下，当LIFO内部Buffer个数到达LIFO溢出水线时会向DDR溢出Buffer，溢出Buffer受总线延时反压，若总线延时过大会反压释放Buffer接口，若溢出Buffer个数大于DDR空间可溢出Buffer数，BMU会在DDR空间真满后丢弃Buffer。当LIFO内部Buffer个数到达LIFO回读水线时会从DDR回读Buffer，回读Buffer受总线延时反压，当LIFO空时，向BMU申请Buffer，会返回全0的Buffer。

软件管理模式，BMU内部注入等同于释放Buffer，注入方式和“**释放Buffer**”节释放Buffer相同；删除等同于申请Buffer，删除方式请参考“**申请Buffer**”节。

1.5.1.8 QoS 反压

对PPE，支持对32个逻辑Pool，每个逻辑Pool的9个QoS单独配置双门限丢弃水线，水线单位粒度为64个Buffer，QoS反压的产生和撤销需要检查逻辑Pool和物理Pool的资源情况，当逻辑Pool的Buffer总数小于其设置的QoS低水线产生反压指示，指示接口模块（PPE）丢弃对应报文；当逻辑Pool的Buffer总数大于其设置的QoS高水线，撤销反压指示接口模块（PPE）的指示，采用写消息反压PPE。

须知

Hi309a中PPE侧仅接收逻辑Pool0~15的QoS反压消息。

写反压消息数据结构：

BMU到PPE的QoS写反压消息被分为3次128bit位宽的burst1写操作，对应结果如图1-3所示。其中每个有效word中0~15bit中每1bit对应到是否QoS是否反压对应逻辑Pool。其中1表示反压，0bit表示解压反压。Hi309a中PPE侧仅接收逻辑Pool0~15的QoS反压消息，每个word中高16bit反压消息无效。

图 1-3 写反压数据结构

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
写操作0	对应逻辑Pool																																
	Word0 QoS0	no use																															
	Word1 QoS1	no use																															
	Word2 QoS2	no use																															
写操作1	Word3 QoS3	no use																															
	Word0 QoS4	no use																															
	Word1 QoS5	no use																															
	Word2 QoS6	no use																															
写操作2	Word3 QoS7	no use																															
	Word0 QoS8	no use																															
	Word1	rsv32bit																															
	Word2	rsv32bit																															
	Word3	rsv32bit																															



须知

BMU的QoS反压非绝对准确，因为在发生block在逻辑Pool和物理Pool之间迁移以及Buffer在LIFO和逻辑Pool之间溢出回读时，会导致剩余Buffer计数更新不准确，产生误反压。

在BMU工作时如果要关闭写消息反压，需要先关闭PPE的QOS接收使能，再关闭BMU的QOS发送使能。

1.5.2 多模及虚拟化

匹配Hi309a芯片的规格，BMU支持0~4一共5个VFID，每个VFID可以对应BMU的一个或多个物理Pool（缓存资源池）。

- BMU提供32个逻辑Pool，系统可对VF使用的逻辑Pool进行分配管理。
- BMU的多模隔离下，各个逻辑Pool可以单独使能、各个物理Pool可以单独初始化，相互间没有依赖。
- BMU上报中断时将伴随VFID信息（每个VF/PF有单独的中断线送往GIC），并提供异常逻辑Pool ID，Hypervisor或GIC可根据VFID送给相应VM。

表 1-2 Timer 中断产生时序图

软件看到的VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
PF	芯片级公共	-	0x0000	芯片公共寄存器	寄存器地址都有实体寄存器与之对应。
		-	0x0200	全局中断寄存器	
		-	0x0400	全局公共配置寄存器	
		-	0x1700	全局状态上报寄存器	



软件看到的 VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
	PF_VF0	-	0x13000	PF访问该VF的中断	1. 逻辑Pool相关的寄存器只有地址，需要映射到“业务级_PF”实体寄存器。 2. 其余寄存器地址都有实体寄存器与之对应。 3. 该组寄存器地址中每个偏移地址的含义与“业务级_PF”相应偏移地址含义相同，只是对应VF0。
		-	0x13300	PF访问该VF的业务配置	
		-	0x14800	PF访问该VF的状态、业务维测	
	PF_VF1	-	0x23000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF1。
		-	0x23300	PF访问该VF的业务配置	
		-	0x24800	PF访问该VF的状态、业务维测	
	PF_VF2	-	0x33000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF2。
		-	0x33300	PF访问该VF的业务配置	
		-	0x34800	PF访问该VF的状态、业务维测	
	PF_VF3	-	0x43000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF3。
		-	0x43300	PF访问该VF的业务配置	
		-	0x44800	PF访问该VF的状态、业务维测	



软件看到的 VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
	PF_VF4	-	0x53000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF4。
		-	0x53300	PF访问该VF的业务配置	
		-	0x54800	PF访问该VF的状态、业务维测	
	PF_VF5 (未使用)	-	0x63000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF5。
		-	0x63300	PF访问该VF的业务配置	
		-	0x64800	PF访问该VF的状态、业务维测	
	PF_VF6 (未使用)	-	0x73000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF6。
		-	0x73300	PF访问该VF的业务配置	
		-	0x74800	PF访问该VF的状态、业务维测	
	PF_VF7 (未使用)	-	0x83000	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF7。
		-	0x83300	PF访问该VF的业务配置	
		-	0x84800	PF访问该VF的状态、业务维测	
VF0	业务级_VF0	64kByte	0x180000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。



软件看到的 VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
			0x183000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x183300	该VF处理业务时的配置	
			0x184800	该VF处理业务时的维测统计	
			0x185000	该VF处理业务时的命令	
VF1	业务级_VF1	64kByte	0x190000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。
			0x193000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x193300	该VF处理业务时的配置	
			0x194800	该VF处理业务时的维测统计	
			0x195000	该VF处理业务时的命令	
VF2	业务级_VF2	64kByte	0x1A0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。



软件看到的 VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
			0x1A3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1A3300	该VF处理业务时的配置	
			0x1A4800	该VF处理业务时的维测统计	
			0x1A5000	该VF处理业务时的命令	
VF3	业务级_VF3	64kByte	0x1B0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。
			0x1B3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1B3300	该VF处理业务时的配置	
			0x1B4800	该VF处理业务时的维测统计	
			0x1B5000	该VF处理业务时的命令	
VF4	业务级_VF4	64kByte	0x1C0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。



软件看到的 VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
			0x1C3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1C3300	该VF处理业务时的配置	
			0x1C4800	该VF处理业务时的维测统计	
			0x1C5000	该VF处理业务时的命令	
VF5	业务级_VF5 (未使用)	64kByte	0x1D0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射。
			0x1D3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1D3300	该VF处理业务时的配置	
			0x1D4800	该VF处理业务时的维测统计	
			0x1D5000	该VF处理业务时的命令	
VF6	业务级_VF6 (未使用)	64kByte	0x1E0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射



软件看到的VF/PF	模块寄存器地址接口	大小	起始地址	说明	备注
			0x1E3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1E3300	该VF处理业务时的配置	
			0x1E4800	该VF处理业务时的维测统计	
			0x1E5000	该VF处理业务时的命令	
VF7	业务级_VF7（未使用）	64kByte	0x1F0000	芯片级寄存器（该VF只读）	该组寄存器是“芯片级”这组寄存器的重映射
			0x1F3000	该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射。 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同。
			0x1F3300	该VF处理业务时的配置	
			0x1F4800	该VF处理业务时的维测统计	
			0x1F5000	该VF处理业务时的命令	

BMU寄存器地址空间详细划分如下：

PF占用64KB*9地址空间，其中芯片公共寄存器包含设备号寄存器、芯片版本寄存器、加速器能力寄存器和SR-IOV寄存器；全局中断寄存器包含PF的中断使能寄存器、中断状态寄存器、中断注入寄存器和中断信息寄存器；全局公共配置寄存器包含制式间共享资源分配寄存器、Memory时序和功耗调节寄存器及其他制式共用寄存器；全局状态上报寄存器包含所有DFX状态和统计寄存器；PF下的PF_VF0~PF_VF4业务处理寄存器供PF访问VF资源使用。

VF做业务时使用VF各自的业务级VF寄存器空间，分别占用64KB地址空间，VF空间内的寄存器相对于VF的基地址的相对偏移与PF对应的寄存器相对于PF基地址的相对偏移相同，但VF只能看到虚拟逻辑Pool和虚拟物理Pool。



图 1-4 VF 资源映射 a

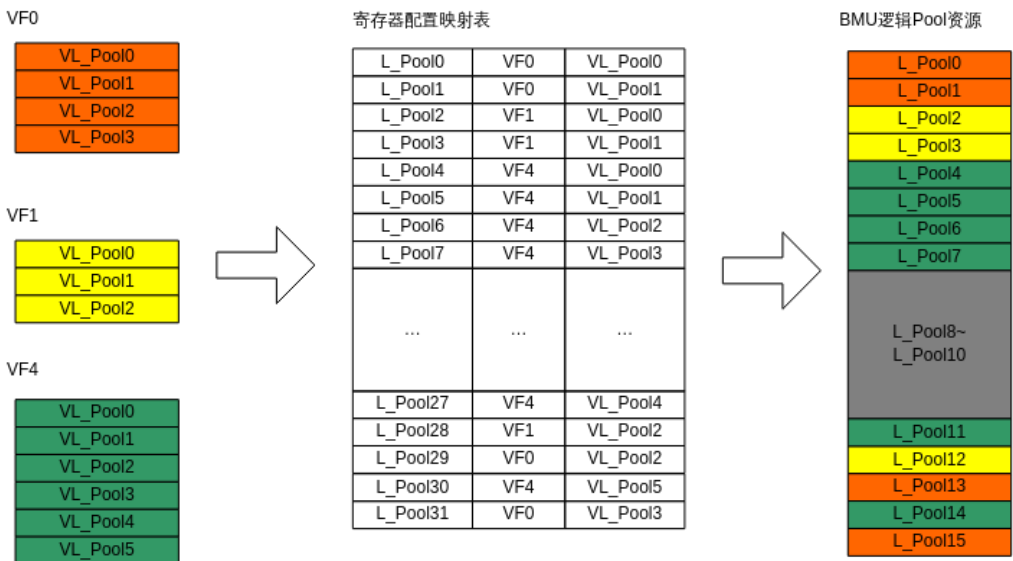
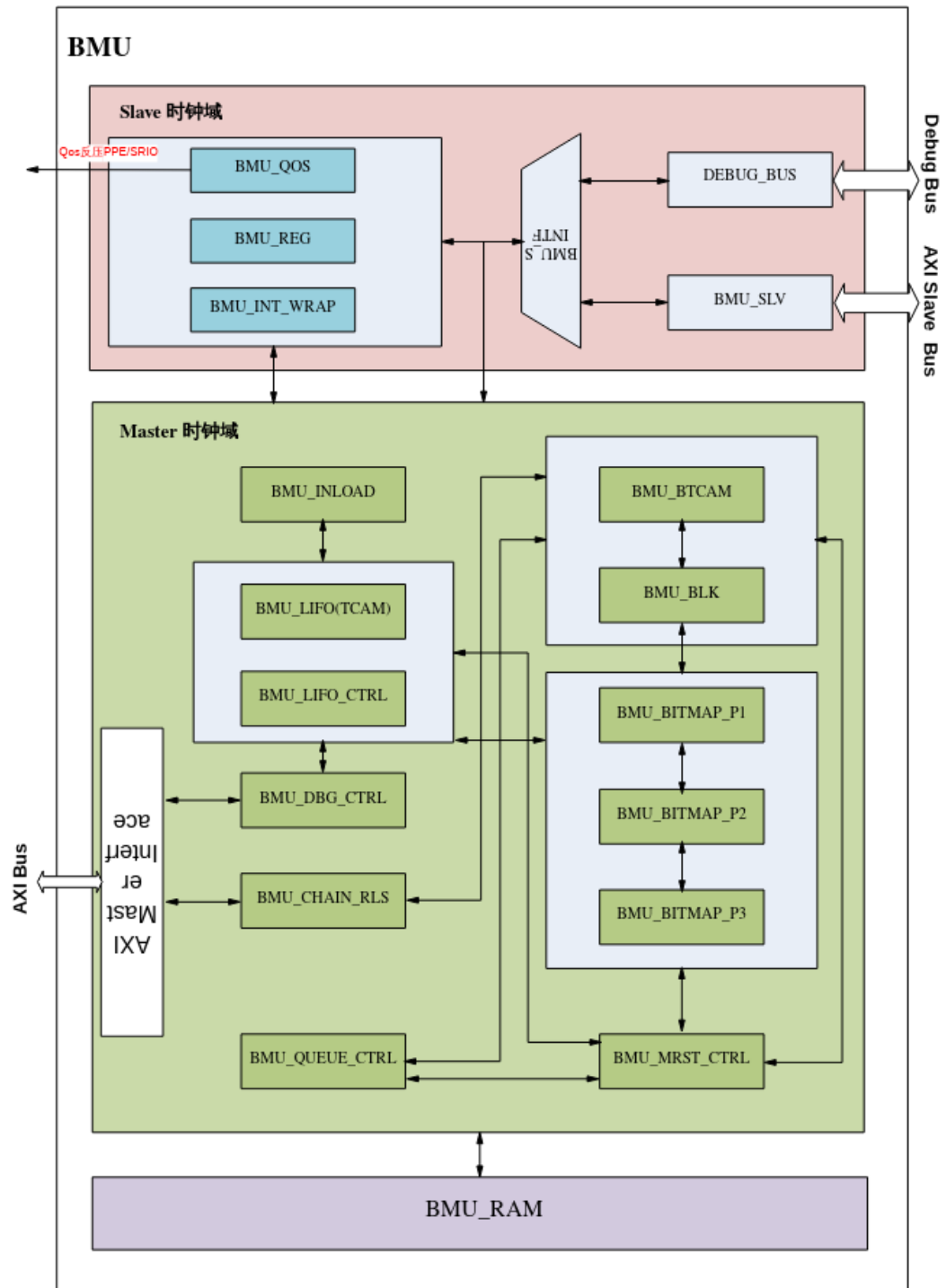


图 1-5 VF 资源映射 b



软件配置BMU_LP2VLP_MAP和BMU_PP2VPP_MAP分别实现虚拟逻辑Pool到逻辑Pool和虚拟物理Pool到物理Pool的映射关系划分，当VF使用业务级VF寄存器操作BMU时，BMU硬件按照映射关系表自动完成虚拟逻辑Pool到逻辑Pool和虚拟物理Pool到物理Pool的转换，转化到相应逻辑Pool和物理Pool 操作。

支持多模复位，支持软件按VF复位BMU，BMU自动完成该VF的各个逻辑Pool和物理Pool复位，回收当前复位制式所占用BMU共享资源（包含LIFO、TCAM、Bitmap



等），从而实现支持单制式复位，复位后物理Pool和逻辑Pool均清空，需要软件重新初始化。

须知

1. 针对软件的多模异常访问，BMU硬件只保证不挂死，不能保证返回有效值或软件期待值，BMU内部做如下处理：
对BMU内部资源和寄存器有破坏性的访问操作，比如申请释放Buffer、注入删除Block、写寄存器操作，直接丢弃，并上报非法访问中断；
对BMU内部资源和寄存器没有破坏性的访问操作，比如读寄存器操作，返回一个无效随机值，并上报非法访问中断。
2. 对0x5000~0xFFFFC地址除申请释放Buffer，注入删除Block地址外，其余地址禁止访问，可能出现不可预知错误。

1.5.3 缓存跟踪

BMU模块支持缓存跟踪。

申请Buffer时，BMU支持增加维测信息到缓存块；释放Buffer时，BMU可以更新维测信息。

维测信息需要包括SRC_ID、LPID、时间戳、Buffer地址、逻辑Pool ID、软件自定义字段（UDF）等信息。

维测信息内容并不用于硬件检测，主要用于软件的后台检查（如通过时间戳检查是否长时间不释放），以及中断后启动的分析与检查。

启用缓存跟踪功能将增加对DDR的读写开销，影响系统性能。该功能可按逻辑Pool配置开关。申请/释放时的维测信息更新操作支持分别使能。

须知

维测信息由于总线阻塞会出现概率性丢失，软件可配置是否开启反压使能，保证不丢失；同时由于总线延迟存在维测信息的更新时延。



图 1-6 报文存储结构



1.5.4 硬件魔术字

BMU在硬件管理模式下支持魔术字，Block申请时填写魔术字，Block还回时对魔术字做校验，比较该Block下的所有Buffer有没有被踩踏，若踩踏上报中断并记录被踩踏Buffer地址Block基地址地址，所有Buffer都必须经过校验才被还回物理Pool，校验通过还回物理Pool，校验不通过可以配置为删除 Block或还回Block到物理Pool；

Magic_num填写在维测信息首个Word，Magic_num由寄存器静态配置，是否注入或校验魔术字有单独的bypass开关。

1.5.5 内部时间戳

BMU支持内部时间戳，该时间戳计数不依赖晶振时间戳，用BMU自己时钟计数，精度为1s，原理为每个BMU cycle累加对应的（1000/时钟频率），当计数累加大于10^9时时戳加1；

假设BMU时钟频率600MHz，每个BMU cycle 要累加1.67，硬件不支持小数加法，内部便产生4组计数器，计数器A每cycle加1，计数器B每2cycle加1，计数器C每4cycle加1，计数器D每100cycle加17，A+B+C+D>=10^9后时戳加1；

A、B、C、D四个计数器的累加值由寄存器BMU_TIMER_CFG、BMU_TIMER_FRACTION决定，该值根据实际工作频率换算，公式为：

timer_integer+ timer_half*0.5+ timer_quarter*0.25+ timer_numer/(4* timer_denomin) = 1GHz/BMU时钟频率。



1.5.6 预取 FIFO

申请接口与LIFO桥接模块，内部给每个逻辑Pool分16深的FIFO，该FIFO支持Bypass和深度可配，当非Bypass时，接口从FIFO申请Buffer，FIFO不满时从LIFO申请Buffer填入FIFO，用于破坏局部LIFO特性以加强写维测性能，破坏强度小于配置深度。

1.6 编程接口

1.6.1 Buffer 指针结构

向BMU申请Buffer就是从BMU指定逻辑Pool获得指向Buffer首地址的指针；向BMU释放Buffer就是将指向Buffer首地址的指针写回BMU对应逻辑Pool中。

须知

- BMU管理的Buffer地址可以是PA、IPA或VA，取决于SMMU页表的配置。同一制式的物理Pool管理的Buffer地址（PA、IPA、或VA）不能重叠。
- 软件注入给BMU的Block必须是连续的4M或8M内存空间（无论是PA、IPA、或VA都必须连续且对齐），而加速器从BMU申请的Buffer经过SMMU映射后可能是不连续的PA。

向逻辑Pool释放Buffer时，通过AXI slave写向BMU寄存器的数据结构是：

{ Rel, Rsv, Addr[47: 0]};

表 1-3 BMU 释放缓存数据结构

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6		4	3	2	1	0										
Addr[31:0]																															
Rel	Rsv															Addr[47:33]															
说明																															
Addr		报文接收描述符地址的47~0位，低6位应当为全0（按照Cache Line长度共64Byte对齐）。																													
Rsv		保留域。																													
Rel		释放标志位，指示释放方式（注：只有当PD的扩展TLV指示为PPE发送描述符时，回写标志才有效；否则为不回写）																													
		0b00：只需要释放addr对应的单Buffer，不需要读描述符，不需要回写状态；																													



3	3	2	2	2	2	2	2	2	2	1	1	1	1	15	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6		4	3	2	1	0								
						<p>0b01: 需要释放描述符和报文的Buffer，根据描述符指示位决定是否回写状态（当PPE发送方向指示为回写时，不释放发送描述符首片节点的Buffer，其余节点的Buffer都需要正常释放）。</p> <p>注意：</p> <p>1：如果wb=1，且为非分离式Buffer（发送描述符首片和报文在一个Buffer），则不能释放。</p> <p>2：如果wb=0，且为非分离式Buffer（发送描述符首片和报文在一个Buffer），释放时只能释放1次，要避免描述符和报文在同一Buffer被重复释放两次。</p>																							
						<p>0b10: 只需要释放报文的Buffer，不释放描述符Buffer，根据描述符指示位决定是否回写状态。</p> <p>注意：如果wb=1，且为非分离式Buffer（发送描述符首片和报文在一个Buffer），则该Buffer不能释放。</p>																							
						<p>0b11: 不释放，根据描述符指示位决定是否回写状态。</p>																							

1.6.2 注入 block 数据结构

{ Rsv, Addr[47: 0]}

1.6.3 维测信息

图 1-7 维测信息

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
VLPID					MAGIC_NUM																												
VLPID1					LPID1					SRC_ID1															FLG1		Rsv		UDF1[4:0]				
VLPID0					LPID0					SRC_ID0															FLG0		Rsv		UDF0[4:0]				
Times																																	
USER_DATA																																	
USER_DATA																																	
USER_DATA																																	
USER_DATA																																	

表 1-4 维测信息数据域说明

数据域	说明
MAIGC_NUM[26:0]	魔术字, Block从物理Pool到逻辑Pool时更新。
VLPID[4:0]	虚拟Pool ID, Block从物理Pool到逻辑Pool时更新。
Times[31:0]	全局时间戳, 只在申请时才更新。
VLPID0/VLPID1[4:0]	申请/释放的虚拟Pool ID。
LPID0/LPID1[2:0]	此处的LPID不是逻辑Pool ID, 指的是申请/释放的LPID。



数据域	说明
SRC_ID0/SRC_ID1[16:0]	申请/释放的SRCID。
FLG0	申请时更新为0b1。
FLG1	释放时更新为0b1，且申请时将其更新为0b0。
UDF0[4:0]	软件由申请Buffer时的地址传入，硬件加速器申请/释放时填充全0。
UDF1[4:0]	软件由释放Buffer时的地址传入，硬件加速器申请/释放时填充全0。
RSV	保留域段。

BMU支持在硬件管理模式写魔术字和软件管理模式初始化注入Buffer时更新Buffer维测信息的Word0，申请Buffer成功和释放Buffer成功后向DDR更新该Buffer维测信息Word1-Word3的操作，维测信息不清除；数据结构如所示，其中灰色的部分BMU可写入（跟随总体数据结构刷新）：

维测信息在报文存储的前64Byte，大小固定为64Byte，不可配置，其中前4个word由硬件更新。

BMU提供时间戳1s的计数精度，计时最大4G秒。

维测信息中的UDF域的低5位UDF0/UDF1[4:0]以及CPU申请释放Buffer的接口地址上的UDF的低5位，软件默认为VCPU的编号。

1.6.4 性能和功耗配置

1.6.4.1 性能优化配置

1.6.4.1.1 链式 Buffer 相关功能配置约束

- 空闲队列溢出水线（寄存器BMU_CHAIN_RLS_FL_WL_CFG[fl_overflow_thd]）：空闲队列的每个节点表示内部深度1024的RAM当前可用的存储空间的地址；当空闲队列的节点个数小于等于设置的空闲队列溢出水线时，表示内部RAM可用空间不足，此时需要把内部RAM的数据溢出到DDR暂时存储起来，待内部RAM有足够空间时再从DDR回读回来。该水线只是某个VF队列溢出的条件之一；
- VF队列溢出水线（寄存器BMU_BUF_RLS_THD_CFG[vf_overflow_thd]）：VF队列的每个节点表示内部深度1024的RAM中存储的对应该VF的数据的地址，当对应VF队列的节点个数大于等于该VF队列配置的水线时，对应VF队列的节点将溢出到DDR暂时存储起来，待内部RAM有足够空间时再从DDR回读回来。该水线只是某个VF队列溢出的条件之一；
- 溢出举例：VF0的溢出水线（寄存器BMU_BUF_RLS_THD_CFG[vf_overflow_thd]）=800，空闲队列溢出水线（寄存器BMU_CHAIN_RLS_FL_WL_CFG[fl_overflow_thd]）=64；如果此时满足：（VF0的队列节点计数>=800）且（空闲队列节点计数<=64）且（DDR中VF0的空间还有空闲），则VF0队列将发生溢出。
- 空闲队列回读水线（寄存器BMU_CHAIN_RLS_FL_WL_CFG[fl_readback_thd]）：空闲队列的每个节点表示内部深度1024的RAM当前可用的存储空间的地址；当空



闲队列的节点个数大于等于设置的空闲队列回读水线时，表示内部RAM可用空间充足，此时需要把以前溢出到DDR中的数据回读到内部RAM。该水线只是某个VF队列回读的条件之一；

- VF队列回读水线（寄存器BMU_BUF_RLS_THD_CFG [vf_readback_thd]）：VF队列的每个节点表示内部深度1024的RAM中存储的对应该VF的数据的地址，当对应VF队列的节点个数小于等于该VF队列配置的水线时，DDR中对应该VF的数据可以回读到内部RAM。该水线只是某个VF队列回读的条件之一；
- 回读举例：VF0的回读水线（寄存器BMU_BUF_RLS_THD_CFG [vf_readback_thd]）=100，空闲队列回读水线（寄存器BMU_CHAIN_RLS_FL_WL_CFG [fl_readback_thd]）=64；如果此时满足：（VF0的队列节点计数≤100）且（空闲队列节点计数≥64）且（DDR中有VF0的数据），则VF0将发生回读。
- 空闲队列反压水线（寄存器BMU_BUF_RLS_FL_INST_BP_CFG [bmu_fl_inst_chain_gap_cfg]）：空闲队列的每个节点表示内部深度1024的RAM当前可用的存储空间的地址；当空闲队列的节点个数小于等于设置的空闲队列反压水线时，链式BUF模块将反压该模块的输入接口，以阻止任何VF数据进入。
- VF队列反压水线（寄存器BMU_INST_CHAIN_GAP_CFG [bmu_inst_chain_gap_cfg]）：VF队列的每个节点表示内部深度1024的RAM中存储的对应该VF的数据的地址，当对应VF队列的节点个数大于等于该VF队列配置的反压水线时，链式BUF模块将反压该模块的输入接口，以阻止该VF数据进入。

须知

为了获得较好的性能，建议：

1. VF队列：反压水线>溢出水线>回读水线>最小资源保证。
2. 空闲队列：反压水线<溢出水线<回读水线。

1.6.4.1.2 寄存器 BMU_POOL_BLOCK_NUM_P 的配置约束

寄存器BMU_POOL_BLOCK_NUM_P[bmu_pool_min_block_num]对应逻辑Pool的最小空间保证；寄存器BMU_POOL_BLOCK_NUM_P[bmu_pool_max_block_num]对应逻辑Pool的最大空间限制。每个逻辑Pool各自拥有独立的配置寄存器，下面以逻辑Pool 0对应的BMU_POOL_BLOCK_NUM_P为例说明其含义：

最大空间限制：限定逻辑Pool 0能够占用物理Pool的Block数的上限，逻辑Pool 0占用的Block个数必然小于等于该配置值。

1. 最小空间保证：限定物理Pool为逻辑Pool 0提前预留的Block个数，保证其独享性，应对未来的可能的业务突发；对于逻辑Pool 0的Block占用没有任何约束关系和因果关系，不等同于逻辑Pool至少占有该配置值的个数的Block。
2. 只限制与逻辑Pool 0共享同一个物理Pool的其他逻辑Pool是否能够申请新的Block，物理Pool必须在满足为该逻辑Pool0预留最小空间保证个数的Block后，还有共享的空闲Block的前提下，才可能分配Block给其他逻辑Pool。

寄存器BMU_POOL_BLOCK_NUM_P配置约束：

逻辑Pool的最小Block数BMU_POOL_BLOCK_NUM_P[bmu_pool_min_block_num]应该为逻辑Pool独享的Block个数，建议，如不使用可以配置为0或者兼容定长Pool模式配置为0；



归属于同一物理Pool的所有逻辑Pool的逻辑Pool的最小Block数之和小于等于该物理Pool的注入Block个数；

逻辑Pool的最大Block数BMU_POOL_BLOCK_NUM_P[bmu_pool_max_block_num]应该为满足逻辑Pool最大业务能力的Block个数，且小于等于物理Pool注入的Block个数。

须知

在变长Pool共享场景下，要求逻辑Pool的BMU_POOL_BLOCK_NUM_P【bmu_pool_max_block_num】配置为2以上，以满足逻辑Pool能够向物理Pool释放Block，兼容定长Pool模式不受此限制。

1.7 应用配置

1.7.1 软件指南

1.7.1.1 Buffer 碎片化回收

Buffer碎片化回收用于系统长时间运行后BMU内部Buffer碎片化导致大部分Block不能回收的问题，通过调节LIFO和逻辑Pool水线将在逻辑Pool内碎片化的Buffer回收回物理Pool使物理Pool中的Block最大化被回收，步骤如下：

- 步骤1** 检查要回收的物理Pool与逻辑Pool的绑定关系，将系统运行中的对应逻辑Pool的Buffer最大化释放回BMU，调低其他HAC的申请释放水线，使在HAC中存留的Buffer数达到最少。
- 步骤2** 检查逻辑Pool是否使能预取FIFO，若打开，将BMU_LP_PRE_FIFO_CFG的bit11配置为1，排空预取FIFO，查询BMU_PRE_BUF_NUM_P，待其值为0，表示预取FIFO已排空，进行**步骤3**。
- 步骤3** 配置对应逻辑Pool LIFO的溢出回读水线寄存器BMU_WL_P，将申请水线配置为1，溢出水线配置为4；配置逻辑Pool申请Block水线寄存器BMU_POOL_BLOCK_REQ_WL_P为1，释放Block水线寄存器BMU_POOL_BLOCK_RLS_WL_P为一个Block的Buffer数减一。

等待BMU内部将逻辑Pool中的Block还回物理Pool：若打开魔术字校验，需要等待逻辑Pool中BMU_LP_BLK_CNT值*一个block魔术字校验时间，所有的模式公共配置寄存器不允许动态配置，除非所有VF都复位。且逻辑Pool使能只允许在复位前后动态开关。在BMU初始化前需要配置GIC的中断映射表和SMMU映射表。

----结束

1.7.1.2 BMU PF 初始化

- 步骤1** 在SubCtrl寄存器中配置SC_BMU_ICG_DIS为0b11，配置SC_BMU_RESET_REQ为0b11；等待1ms配置SC_BMU_RESET_DREQ为0b11，配置SC_BMU_ICG_EN为0b11；读取SC_BMU_CLK_ST为0b11，SC_BMU_RESET_ST都为0b00后表示BMU时钟复位正常，可进行寄存器配置。



- 步骤2** 芯片上电后，软件配置BMU的memory上电初始化使能寄存器，BMU_MEM_INIT_ENABLE，开启BMU的内部Memory自动初始化操作，软件轮询BMU_MEM_INIT_DONE寄存器是否为1，为1标志设备初始化完成，进入**步骤3**。
- 步骤3** 根据需要配置动态clock gating使能配置寄存器BMU_CLK_DYNAMIC_GATE_EN为1，并配置BMU_GATING_WINDOW寄存器为非0值。
- 步骤4** 配置BMU_TCAM_SCAN_CFG寄存器为1，BMU_ECC_CHECK_EN寄存器为1。
- 步骤5** 配置芯片公共寄存器SR_IOV_CTL、SR_IOV_VF_OFFSET。
- 步骤6** 配置BMU的中断使能寄存器：BMU_COM_INT_ENABLE，将系统关心的中断使能控制位配置为1。
- 步骤7** 配置逻辑Pool管理模式选择寄存器BMU_POOL_INIT_RDY[bmu_lp_mng_mode_sel]，选择软件管理模式或硬件管理模式。

----结束

若为软件管理模式：

- 步骤1** 配置逻辑Pool存储模式选择寄存器BMU_POOL_INIT_RDY[bmu_lp_store_mode_sel]，选择软件管理模式下的存储模式。
- 步骤2** 配置软件管理模式LIFO存储行为下每个逻辑Pool的BMU_LP_OVF_DDR_SPACE，BMU_LP_OVF_DDR_ADDR，配置溢出空间和地址。
- 步骤3** 配置软件管理模式下每个逻辑Pool的BMU_SB_LP_BUFFER_LADDR、BMU_SB_LP_BUFFER_HADDR，配置每个逻辑Pool的地址上下边界。
- 软件管理模式配置完成。
- 步骤4** 配置每个逻辑Pool LIFO基地址寄存器BMU_LIFO_BADDR_P，LIFO大小寄存器BMU_SIZE_P，BMU_SIZE_P在需要工作的逻辑Pool下禁止配置为0，在非工作的逻辑Pool下禁止配置为非0；LIFO溢出回读水线寄存器BMU_WL_P。
- 步骤5** 根据需要配置逻辑Pool的LIFO假空时，返回给CPU的值BMU_EMPTY_DATA；根据需要配置逻辑Pool多模复位或者BMU独立复位时，返回给CPU的值BMU_RST_BP_DATA；配置LIFO溢出回读Buffer个数寄存器BMU_OVRB_BUFFER_NUM_P；配置Block扫描配置寄存器BMU_BLK_SCAN_CFG。
- 步骤6** 配置链式Buffer释放的空闲队列配置寄存器BMU_CHAIN_RLS_FL_WL_CFG、BMU_BUF_RLS_FL_INST_BP_CFG、BMU_CHAIN_OVER_LENGTH_THD。
- 步骤7** 配置维测信息反压配置寄存器BMU_TAG_BP_CFG。
- 步骤8** 根据需要配置状态机超时门限寄存器BMU_FSM_HBEAT_THLD、BMU_MRST_FSM_HBEAT_THLD。
- 步骤9** 配置魔术字配置寄存器BMU_MAGIC_NUMBER。
- 步骤10** 配置全局控制寄存器。
1. 配置BMU_VF_MANAGE_EABLE。
 2. 配置BMU_VF_STREAM_ID。
 3. 配置BMU_LP2PP_MAP。
 4. 配置BMU_LP2VLP_MAP。
 5. 配置BMU_PP2VPP_MAP。



6. 配置BMU_AXI_QOS_CACHE_CFG。
7. 配置BMU_DATA_ARUSER_CFG0~1。
8. 配置BMU_BD_ARUSER_CFG0~1。
9. 配置BMU_DATA_AWUSER_CFG0~1。
10. 配置BMU_BD_AWUSER_CFG0~1。
11. 配置BMU_OUTST_DEPTH_CFG。
12. 配置BMU_TIMER_CFG。
13. 配置BMU_TIMER_FRACTION。

步骤11 配置BMU_CTRL寄存器，其中[bmu_enable]配置为1，根据需要配置BMU_CTRL[bmu_mng_mode_cfg]，其他域段如[bmu_endian_cfg]等根据需要配置。

----结束

须知

当BMU的寄存器BMU_SIZE_P、BMU_WL_P和BMU_OVRB_BUFFER_NUM_P配置不合理时会导致BMU的溢出回读震荡，BMU_POOL_BLOCK_REQ_WL_P和BMU_POOL_BLOCK_RLS_WL_P配置不合理时会导致BMU的block申请释放震荡；除了碎片化回收场景按照约束步骤配置震荡外，其他场景禁止配置震荡，因为可能会导致：申请释放Buffer效率降低，功耗增加，频繁上报lifo假空中断、软件管理模式的假满中断，频繁反压、PPE，频繁访问DDR或内部bitmap等行为。

1.7.1.3 VF 业务初始化

步骤如下（如果启动多个VF业务，则重复执行此流程）：

步骤1 配置VF各自的业务配置寄存器：

1. 配置BMU_ENDIAN_CFG。
2. 配置BMU_INST_CHAIN_GAP_CFG。
3. 配置BMU_BUF_RLS_THD_CFG。
4. 配置BMU_BUF_RLS_DDR_BASE。
5. 配置BMU_BUF_RLS_DDR_DEPTH。

步骤2 配置BMU的中断使能寄存器BMU_INT_ENABLE，将系统关心的中断使能控制位配置为1。

步骤3 配置逻辑Pool管理的Buffer长度寄存器BMU_POOL_BUFFER_SIZE_P，逻辑Pool最大空间限制和最小空间保留寄存器BMU_POOL_BLOCK_NUM_P，逻辑Pool申请释放Block水线寄存器BMU_POOL_BLOCK_REQ_WL_P和BMU_POOL_BLOCK_RLS_WL_P。

步骤4 根据需要配置QoS水线寄存器BMU_PPE_QOS_WL、BMU_ADD_PPE_TIME_CFG、BMU_ADD_PPE_QOS0_ADDR、BMU_ADD_PPE_QOS8_ADDR、BMU_ADD_PPE_AWUSER_CFG、BMU_ADD_PPE_STREAM_ID。

步骤5 硬件管理模式软件向BMU该VF下的各物理Pool注入Block。

步骤6 软件将需要物理Pool管理的所有Block注入完毕后，物理Pool初始化即完成。



- 步骤7** 配置逻辑Pool使能寄存器BMU_POOL_INIT_RDY[bmu_pool_en]为1，使能指定的逻辑Pool；寄存器BMU_POOL_INIT_RDY的其他域段[bmu_buf_64b_rev_en]、[bmu_st_tag_en]、[bmu_ld_tag_en]根据具体应用需求进行配置。
- 步骤8** 硬件自动完成开启使能的逻辑Pool的初始化，逻辑Pool自动获取物理Pool的Block资源，逻辑Pool的LIFO通过回读自动完成Buffer填充工作。
- 步骤9** 软件检查逻辑Pool初始化完成寄存器BMU_POOL_INIT_RDY[bmu_pool_init_rdy]，若为0b1，表示当前逻辑Pool初始化完成，当VF的所有逻辑Pool均已初始化完成，则VF初始化完成。
- 步骤10** 软件管理模式下向逻辑Pool注入Buffer，注入完成后查询对应的Buffer计数器BMU_SB_BUF_NUM_P，该寄存器值为注入Buffer的个数，该计数器等于预期的Buffer数时注入完成。
- 步骤11** 等待PPE，完成初始化后配置BMU_QOS_EN，开启反压和PPE的使能。

----结束

以上步骤完成后，表示BMU初始化完成，PPE等组件可以开始向BMU申请/释放缓存。

须知

- 配置BMU逻辑Pool使能寄存器必须在物理Pool初始化完成之后，否则可能造成异常。
- 配置BMU逻辑Pool使能寄存器后，软件必须读取BMU_POOL_INIT_RDY，确定逻辑Pool初始化完成，否则可能出现缓存泄漏情况。
- 由于PPE等启动后会立即向BMU申请Buffer指针，因此需要BMU初始化配置完成后，系统再启动PPE等；否则会出现BMU没有初始化完成而PPE等即开始向BMU申请缓存的情况。

1.7.2 VF 内存迁移流程

BMU支持内存在VF间动态迁移，流程如下：

- 步骤1** 软件查询迁移的源VF的指定物理Pool的空闲Block个数寄存器BMU_PP_BLK_QUEUE_INFOB，若寄存器值非零则执行**步骤2**；否则当前物理Pool无空闲Block动态删除，无法进行内存迁移。
- 步骤2** 软件读取迁移的源VF的指定物理Pool的寄存器BMU_BLK_RADDR_P，执行物理Pool的动态Block删除。
- 步骤3** 软件首先轮询迁移的源VF的删除Block的状态查询寄存器BMU_BLK_DELETE_STATUS[cpu_delete_blk_rdy]为1。
- 若BMU_BLK_DELETE_STATUS[cpu_delete_blk_success]为1，表示已从指定物理Pool成功删除了一个Block，该Block为BMU_BLK_DELETE_STATUS[cpu_deleted_block]，执行**步骤4**。
 - 若BMU_BLK_DELETE_STATUS[cpu_delete_blk_success]为0，表示当前物理Pool删除Block失败，原因是空闲Block已被逻辑Pool申请，迁移流程结束。
- 步骤4** 软件将**步骤3**成功删除的Block写入迁移的目的VF的指定物理Pool的寄存器BMU_BLK_WADDR_P，执行物理Pool的动态Block注入。



步骤5 软件首先轮询迁移的目的VF的注入Block的状态查询寄存器
BMU_BLK_INJECT_STATUS[cpu_inject_blk_rdy]为1。

若 BMU_BLK_INJECT_STATUS[cpu_inject_blk_success]为1，表示已向指定物理Pool成功注入了一个Block，该Block为BMU_BLK_INJECT_STATUS[cpu_inject_block]，迁移流程结束。

----结束

1.7.3 BMU 多模复位的方法

系统对于BMU的复位有2种形式：一种用于单模式（VFID）的复位，此时对该VFID所有的物理Pool和逻辑Pool分别进行复位；另一种通常用于系统异常时，对整个BMU进行独立复位。

BMU按VF多模复位的方法：在多模应用时，BMU需要支持按模式，也就是按VFID进行软复位。按VF复位时先停掉该VF下所有物理Pool和逻辑Pool对应的业务。各CPU是否已经释放缓存，以及各加速器是否已完全释放或丢弃了所有逻辑Pool的缓存，由系统软件判断。

须知

保证BMU能够按模式安全复位的首要前提是在BMU启动复位后，软件和加速器不能再通过总线向BMU申请释放Buffer操作，复位前需软件可查询当前逻辑Pool中的Buffer计数，待计数处于相对稳定时（变化在10以内），复位更安全。

软件可以同时配置多个VF复位，但BMU会在内部对待复位的VF进行轮询，完成一个VF的复位后再复位下一个。

硬件管理模式复位流程：

- 步骤1** 关闭该VF下的所有逻辑Pool使能（BMU_POOL_INIT_RDY[bmu_pool_en]）。
- 步骤2** 配置VF0~4的BMU_VF_RST_EN[bmu_vf_rst_en]为1，对VF进行复位，查询到对应VF的BMU_VF_RST_EN[bmu_vf_rst_rdy]为0时表示BMU已接受对应VF复位请求准备开始复位。
- 步骤3** 等待2ms后查询对应VF复位完成信号BMU_VF_RST_EN[bmu_vf_rst_rdy]为1，确保VF复位完成，进入**步骤5**，若为0，进入**步骤4**。
- 步骤4** 读取偏移地址BMU_CHAIN_RST_RDY寄存器，若对应vf的bit为1，则表示对应VF复位成功。
- 步骤5** 若复位后该VF不需再使用，复位流程到此结束；若复位后需要重新使用该VF业务，则软件继续执行VF业务初始化步骤，参考“**VF业务初始化**”章节。根据需要，可以更改VF的配置寄存器。

须知

注意不能修改VF共用的寄存器，以免影响其他VF的物理Pool和逻辑Pool的正常工作。

----结束



软件管理模式逻辑Pool复位流程：

步骤1 申请走对应逻辑Pool的所有Buffer。

步骤2 待Buffer全部申请走后关闭对应逻辑Pool使能。

步骤3 若复位后逻辑Pool不需再使用，复位流程到此结束；若复位后需要重新使用该逻辑Pool业务，则软件继续执行业务初始化步骤。根据需要，可以更改逻辑Pool的配置寄存器，注意不能修改逻辑Pool共用的寄存器，以免影响其他物理Pool和逻辑Pool的正常工作。

----结束

须知

1. 由于某个VF多模复位时，其他制式可能正在正常工作，复位时延在ms级，按物理Pool最大资源256个Block配置复位时间大约在2ms左右。
2. 所有的单模复位都不会复位配置值寄存器和原始中断寄存器，如果该VF重新被分配，残留的某些寄存器有影响，需要软件在重新分配前将这些寄存器清除或配置正确。
3. 硬件管理模式和软件管理模式业务复位相互独立，可根据需求分别复位。
4. 若软件管理模式与硬件管理模式共存于一个VF，需要先复位软件管理模式再复位硬件管理模式。

1.7.4 中断和中断查询

BMU提供6根中断线，针对一个PF中断和5个VF中断。

部分非关键中断提供两级中断状态查询机制，即首先通过中断信号和原始中断寄存器告知系统BMU当前出现的异常中断类型，然后系统查询对应的状态寄存器确认中断信息寄存器。

BMU中断查询规则：首先根据BMU上报的中断制式，软件读取指定制式的原始中断寄存器，获取中断类型后，读取该中断类型对应该制式的中断信息寄存器。

BMU提供的中断包括LIFO RAM配置中断、内部RAM ECC校验错误中断、逻辑Pool空中断、逻辑Pool内部LIFO空中断、QoS水线全反压中断、重复释放中断、boundary中断、访问未使能逻辑Pool中断、Buffer映射除不尽中断、FIFO空读满写中断、状态机超时中断、总线Error中断、假满反压中断、链式Buffer超长链中断等，所有中断都提供中断使能寄存器。

须知

RAM配置中断需要软件将需要LIFO的size配置完成以后，配置BMU_CTRL全局使能后硬件才能检查各LIFO的size之和是否超过4096，超过报中断。

CPU读取中断状态寄存器时，如果有多次曾经产生过相同的中断状态，则只记录最早一次的异常状态。

BMU提供了多个异常现场保护寄存器，在BMU出现异常时，CPU可以通过查询该寄存器获取更具体的情况。



关键的异常中断上报中断的同时，在中断信息寄存器内部直接提供逻辑Pool ID。具体详见《Hi309aBMU寄存器描述》。

系统可根据VFID判断由哪个处理器（或VF）处理相应的中断。中断状态可索引到逻辑Pool ID一级，不同物理和逻辑Pool会对应不同的VFID，BMU上报中断会区分VFID。

- BMU提供如下的全局异常中断检测，上报中断给PF：
 - BMU中所有逻辑Pool的LIFO的配置空间超过片内LIFO空间，上报错误中断。
 - 当BMU对DDR进行访问操作时，总线反馈非OKAY响应，上报错误中断，并统计错误计数。
 - 内部RAM ECC校验错误中断，统计错误计数，记录错误地址。
 - 内部FIFO空读满写中断，记录异常FIFO ID。
 - 内部状态机超时中断，记录异常状态机ID。
- BMU提供如下的基于逻辑Pool操作的异常中断检测，按制式上报中断到指定VF：
 - 外部模块释放的Buffer指针不在BMU所管理的缓存范围内，该Buffer丢弃，上报boundary错误中断，并记录Buffer+ LPID+SRCID。
 - 外部模块释放的Buffer指针到Bitmap的映射除不尽，该Buffer丢弃，上报除不尽错误中断，并记录Buffer+ LPID+SRCID+逻辑Pool ID。
 - 外部模块释放的Buffer指针多次重复释放，该Buffer丢弃，上报重复释放中断，并记录Buffer+ LPID+SRCID+逻辑Pool ID。
 - 当某个逻辑Pool的LIFO读空时（表示没及时回读），上报逻辑Pool内部LIFO空中断，并记录逻辑Pool ID。
 - 当某个逻辑Pool的所有空间读空时，上报逻辑Pool空中断，并记录逻辑Pool ID。
 - 申请未始能逻辑Pool中断，记录VFID+LPID+SRCID+逻辑Pool ID。
 - 释放未始能逻辑Pool中断，记录VFID+LPID+SRCID+逻辑Pool ID，并记录错误Buffer指针。
 - 逻辑Pool的9个QoS水线全部反压PPE上报QoS全反压中断。

当CPU释放链式Buffer时受到假满反压时上报假满反压中断。

1.8 性能

SRAM的工作频率最高位1000MHz，接口位宽为128bit。

- 对于Normal空间，读写吞吐率共计（最高）为：
 $128\text{bits} \times 1000\text{MHz} = 128\text{Gbps} = 16\text{GB/s}$
- 对于Exclusive空间，数据位宽为64bit，读吞吐率为：
 $64\text{bits} \times 1000\text{MHz} = 64\text{Gbps} = 8\text{GB/s}$
- 对于Exclusive空间，数据位宽为64bit，写操作需要进行读-改-写，带宽减半，写吞吐率为：
 $64\text{bits} \times 1000\text{MHz} \div 2 = 32\text{Gbps} = 4\text{GB/s}$

1.9 DFX 应用指导

无。



1.9.1 故障在线检测

1.9.1.1 BMU 配置异常

- 当BMU的寄存器BMU_SIZE_P、BMU_WL_P和BMU_OVRB_BUFFER_NUM_P配置不合理（比如溢出回读水线间隔个数较小）时会导致BMU的溢出回读震荡。
- BMU_POOL_BLOCK_REQ_WL_P和BMU_POOL_BLOCK_RLS_WL_P配置不合理时会导致BMU的block申请释放震，会导致频繁上报lifo假空中断、软件管理模式的假满中断，频繁反压PPE，频繁访问DDR或内部bitmap等行为。
- 配置BMU逻辑Pool使能寄存器必须在物理Pool初始化完成之后，否则可能造成异常。
- 配置BMU逻辑Pool使能寄存器后，软件必须读取BMU_POOL_INIT_RDY，确定逻辑Pool初始化完成，否则可能出现缓存泄漏情况。
- 由于PPE等启动后会立即向BMU申请Buffer指针，因此需要BMU初始化配置完成后，系统再启动PPE等；否则会出现BMU没有初始化完成而PPE等即开始向BMU申请缓存的情况。

1.9.1.2 典型问题定位

1.9.1.2.1 申请 Buffer 异常

向未使能的逻辑Pool申请Buffer故障，中断上报查询
BMU_INT_STATUS[bmu_pool_unen_rd]，中断信息查询
BMU_POOL_RD_ERR_INFO。

1.9.1.2.2 释放 Buffer 异常

- 硬件管理模式向未使能的逻辑Pool释放Buffer故障，中断上报查询
BMU_INT_STATUS[bmu_pool_unen_wr]，中断信息查询
BMU_POOL_WR_ERR_INFO0、BMU_POOL_WR_ERR_INFO1、
BMU_POOL_WR_ERR_INFO2；
- 外部模块释放的Buffer不在BMU管理的Buffer范围内或软件管理模式释放未使能的逻辑Pool，中断上报查询BMU_INT_STATUS[bmu_bound_err]，中断信息查询
BMU_BOUND_ERR_INFO0、BMU_BOUND_ERR_INFO1、
BMU_BOUND_ERR_INFO2，中断统计查询BMU_BOUND_ERR_CNT；
- BMU检测到重复释放错误故障，中断上报查询
BMU_INT_STATUS[bmu_dupli_st_err]，中断统计查询
BMU_BITMAP_ST_ERR_CNT；
- BMU释放Buffer时地址映射除不尽错误，中断上报查询
BMU_INT_STATUS[bmu_div_err]，中断信息查询BMU_DIV_ERR_INFO0、
BMU_DIV_ERR_INFO1、BMU_DIV_ERR_INFO2；
- BMU释放多播Buffer错误，中断查询BMU_INT_STATUS[bmu_mb_err]，中断信息查询BMU_MB_INFO0、BMU_MB_INFO1、BMU_MB_INFO2。

1.9.1.2.3 链式 Buffer 释放时 PPE 发送方向描述符 OWN 域不为 0xA 中断

PPE发送方向描述符OWN域不为，中断上报查询
BMU_INT_STATUS[bmu_non_soft_chain_rls_err]，中断信息查询
BMU_NON_SOFT_CHAIN_RLS_ERR_INFO0、



BMU_NON_SOFT_CHAIN_RLS_ERR_INFO1、
BMU_NON_SOFT_CHAIN_RLS_ERR_INFO2。

1.9.1.2.4 软件管理模式 Buffer 真满丢弃

软件管理模式注入、释放Buffer发生真满丢弃，中断上报查询
BMU_INT_STATUS[bmu_sb_true_full_drop]，中断信息查询
BMU_SB_TRUE_FULL_INFO0、BMU_SB_TRUE_FULL_ERR_INFO1、
BMU_SB_TRUE_FULL_INFO2，中断统计查询BMU_SB_TRUE_FULL_DROP_CNT。该
异常只在软件配置错误时（如注入Buffer超过BMU管理Buffer范围）才会出现，需检
查软件配置。

1.9.1.2.5 软件管理模式假满的反压

软件管理模式假满上报中断，中断上报查询
BMU_INT_STATUS[bmu_sb_false_full_err]。

1.9.1.2.6 BMU 魔术字校验失败

Block返回给物理pool时，进行魔术字校验。如果校验失败，则上报中断，并查询
BMU_INT_STATUS[bmu_mn_check_fail]。中断信息可通过查询
BMU_MN_CHECK_FAIL_BUFF_ADDR_H、BMU_MN_CHECK_FAIL_BUFF_ADDR_L、
BMU_MN_CHECK_FAIL_BLOCK_ADDR、BMU_MN_CHECK_FAIL_ID获得。

1.9.1.2.7 链式 Buffer 循环链的节点异常

链式Buffer异常循环链的节点个数超过配置门限，上报中断，中断上报查询
BMU_INT_STATUS[bmu_chain_over_length_err]，中断信息查询
BMU_CHAIN_OVER_LENGTH_INFO0、BMU_CHAIN_OVER_LENGTH_INFO1、
BMU_CHAIN_OVER_LENGTH_INFO2。

1.9.1.2.8 链式 Buffer 或分离式 Buffer 受到内部队列假满的反压

CPU释放的链式Buffer或分离式Buffer受到内部队列假满的反压中断，中断上报查询
BMU_INT_STATUS[bmu_false_full_err]。

1.9.1.2.9 逻辑 pool 被读空中断

逻辑pool被读空中断，中断上报查询BMU_INT_STATUS[bmu_all_empty]，中断信息
查询BMU_ALL_EMPTY_INFO。

1.9.1.2.10 逻辑 pool 的内部 SRAM 读空中断

逻辑pool的内部SRAM读空中断，中断上报查询BMU_INT_STATUS[bmu_in_empty]，
中断信息查询BMU_IN_EMPTY_INFO。

1.9.1.2.11 逻辑 POOL 的 Buffer 剩余低于配置的所有 9 个水线

QoS水线全反压中断，中断上报查询BMU_INT_STATUS[bmu_die0_qos_all_bp]/
BMU_INT_STATUS[bmu_die1_qos_all_bp]。

1.9.2 模块维测



1.9.2.1 内部状态查询

表 1-5 内部状态查询

内部状态	状态机状态	BMU FSM STATE0. BMU FSM STATE1
	fifo的空满状态	BMU_FIFO_DFX0, BMU_FIFO_DFX1、BMU_FIFO_DFX2、 BMU_FIFO_DFX3
	lifo预取的空满指示	BMU_LIFO_PRE_FLG0, BMU_LIFO_PRE_FLG1
	BMU_IN_EMPTY_STS	BMU当前各逻辑pool内部LIFO空的实时状态
	BMU_IN_FULL_STS	BMU当前内部各逻辑pool内部LIFO满的实时状态
	BMU_VF_BLK_STS	BMU VF的Block迁移状态
	BMU_BLK_RDBK_STS	BMU逻辑Pool的Block迁移阻塞回读的状态
	BMU_OVFL_STS	BMU溢出状态寄存器
	BMU_RDBK_STS	BMU回读状态寄存器
	BMU_LP_BLK_REQ_STS	BMU逻辑pool申请block的状态
	BMU_LP_BLK_RELEASE_STS	BMU逻辑pool释放block的状态
	BMU_CPU_QOS_BP	BMU CPU的QOS反压实时状态

1.9.2.2 统计

1.9.2.2.1 功能统计

表 1-6 功能统计

buffert相关	BMU_ALLOCATE_BUF_CNT	BMU分配的所有Buffer统计
	BMU_INST_SINGLE_BUF_CNT	BMU回收的所有单Buffer统计
	BMU_RELEASE_BUF_CNT	回收到BMU资源池中的Buffer个数统计
	-	-



	BMU_PTRNUM_P	BMU硬件管理模式下逻辑pool的实时Buffer个数
	BMU_LIFO_PTR_NUM_P	逻辑pool的lifo实时Buffer个数
	BMU_BITMAP_PTR_NUM_P	逻辑pool的bitmap实时Buffer个数
	BMU_SB_BUF_NUM_P	BMU软件管理模式下逻辑pool的实时Buffer个数。
	BMU_LP_DDR_CNT	BMU软件管理模式下溢出到DDR的Buffer数统计
	-	-
	BMU_POOL_REQ_BUFF_CNT_L	BMU逻辑pool累计申请Buffer数
	BMU_POOL_REQ_BUFF_CNT_H	BMU逻辑pool累计申请Buffer数
	BMU_POOL_RLS_BUFF_CNT_L	BMU逻辑pool累计释放Buffer数
	BMU_POOL_RLS_BUFF_CNT_H	BMU逻辑pool累计释放Buffer数
block相关	BMU_LP_BLK_CNT	BMU逻辑pool中占用的Block数
	BMU_BLK_REQ_CNT_P	逻辑Pool的从物理Pool申请的Block资源统计
	BMU_BLK_RLS_CNT_P	逻辑Pool的向物理Pool释放的Block资源统计
	BMU_BLK_INJECT_CNT	VF注入的Block命令个数统计
	BMU_BLK_DELETE_CNT	VF删除的Block命令个数统计
	BMU_LP_BLK_QUEUE_INFO_OA	lp非空block队列头尾指针
	BMU_LP_BLK_QUEUE_INFO_OB	lp非空block队列深度
	BMU_LP_BLK_QUEUE_INFO_1A	lp空block队列头尾指针
	BMU_LP_BLK_QUEUE_INFO_1B	lp空block队列深度
	BMU_LP_BLK_QUEUE_INFO_2A	lp满block队列头尾指针
	BMU_LP_BLK_QUEUE_INFO_2B	lp满block队列深度
	BMU_PP_BLK_QUEUE_INFO_A	pp空闲block队列头尾指针



	BMU_PP_BLK_QUEUE_INFO B	pp空闲block队列深度
	BMU_FL_BLK_QUEUE_INFO A	BMU空闲队列的队列头尾指针
	BMU_FL_BLK_QUEUE_INFO B	BMU空闲队列的队列深度
接口统计	BMU_BUF_RLS_BUF_IN_CNT	入队的每个VF的链式首Buffer统计
	BMU_BUF_RLS_BUF_OUT_CNT	出队的每个VF的Buffer统计
	BMU_MB_SET_CNT	多播set统计
	BMU_MB_RLS_IN_CNT	多播Buffer释放入口统计
	BMU_MB_RLS_OUT_CNT	多播Buffer释放出口统计
异常情况的统计	BMU_TAG_DROP_CNT	由于总线阻塞的丢失计数统计
	BMU_LIFO_EMPTY_CNT	逻辑Pool的LIFO假空计数统计
	BMU_BUS_ERR_CNT0, BMU_BUS_ERR_CNT1	总线出现错误响应统计计数，分读写通道
	BMU_ECC_2BIT_ERR_CNT	Memory_ECC_2bit错误计数统计
	BMU_ECC_1BIT_ERR_CNT	Memory_ECC_1bit错误计数统计
	BMU_BLK_DEPLI_INJECT_CNT	Block重复注入错误次数统计
	BMU_BLK_EMPTY_INJECT_CNT	Block因资源链空导致注入失败错误次数统计
	BMU_DV_ERR_CNT	地址映射除不尽错误次数统计
	BMU_BITMAP_ST_ERR_CNT	重复释放错误次数统计
	BMU_BOUND_ERR_CNT	Boundary错误次数统计

1.9.2.3 CPU 间接访问内部表项 RAM

CPU可以采用间接方式只读访问BMU内部所有表项RAM。

软件可以通过间接地址访问的方式读取BMU的表项内容。软件在寄存器BMU_BITMAP_INDIRECT_ADDR中写入要访问的RAM Entry地址，查询BMU_BITMAP_INDIRECT_ADDR中是否成功写入了需要间接访问的RAM地址后，即可通过读取BMU_BITMAP_INDIRECT_RD_INTF获取对应Entry的内容，最后查询BMU_BITMAP_INDIRECT_READY可判断间接访问的数据是否已成功准备好并放入寄存器BMU_RAM_DATA0~3。



1.9.3 典型定位案例

NA。

1.10 注意事项

无。



2_{TM}

2.1 概述

TM（Traffic Management）流量管理模块。接收CPU Core和加速器添加的包描述符（PD）任务，进行拥塞避免的RED丢弃或Tail丢弃、队列管理，层次化QOS调度、流量整形、流量统计，将需要丢弃包描述符释放Buffer给缓存管理模块（BMU），非丢弃包描述符根据端口配置属性发送给对应加速器。

2.2 关键术语

表 2-1 TM 关键术语描述

术语	描述
Shaping	整形，队列中的包以规定的流量调度出去。
SP+DRR	调度算法可以配置成SP或DRR，或者一部分队列采用SP算法调度，剩余部分队列采用DRR算法调度。
RED	Random Early Detection，随机早期检测。
WRR	Weighted Round-Robin，权重轮询调度算法，基于包个数调度。
DRR	Deficited Round-Robin，赤字轮询调度算法，基于包长字节数调度。
CIR	Committed Information Rate，承诺信息速率。
PIR	Peak Information Rate，峰值流量速率。
CBS	Committed Burst Size，承诺突发尺寸。
PBS	Peak Burst size，峰值突发尺寸。
Trunk	链路聚合。
pps	Packet per second，每秒的报文数，用于表示包速率，包率越高，传送的包个数越多。



术语	描述
bps	Bit per second, 每秒的比特数, 用于表示比特率, 比特率越高, 传送的数据越大。
PD	Packet Descriptor, 包描述符, 一个PD表示一个包任务。
QoS	Quality of Service, 服务质量。

2.3 特性

TM具有以下特性:

- 队列管理:
 - 队列节点以PD为单位, 单节点位宽256bits。
 - CPU Core通过POE由AXI Slave接口添加入TM。
 - 各加速器添加描述符独享寄存器接口, 通过AXI Slave接口添加入TM。
 - 正常转发1024个队列, 每个队列都支持溢出回读DDR, 且溢出回读基地址、缓存空间大小 (0~ (128*2047) 个PD, 配置单位128个PD), 由PF/VF静态且基于队列配置, 溢出回读的stream_id按照队列所属的VF进行选择。
 - 支持溢出回读DDR cache属性由PF统一配置。
 - 支持分片描述符的整体顺序入队检查, 并对1个报文的所有分片描述符进行整体入队或是整体丢弃处理, 最大支持210分片, 且对于同一个添加包的master不支持分片间和分片内的乱序添加。
- 支持统一的I/O、HAC加速器访问接口。
 - TM有与其他加速器 (如POE) 一致的中断寄存器排列方式, 支持两级中断的查询模式。
 - 支持软件查询包含硬件规格信息的寄存器。
 - 支持软件可读写设备号PF_ID, 功能号VF_ID。
 - 公共统计类寄存器支持写清方式, 业务统计类寄存器支持读清方式, 如有个别统计寄存器应用者不希望读清, 在功能描述中枚举。
- 虚拟化, 多模应用, 并采用统一的虚拟化 PF/VF接口。
 - 支持基于1个PF+5个VF的6个模式的寄存器访问 (包含命令寄存器地址、配置维测寄存器地址、memory访问寄存器地址), 其中PF用于管理所有VF的公共功能以及各个VF之间的资源分配, VF用于独立完成业务处理的功能, PF可对所有VF功能进行访问和配置, 而VF只能访问各自资源和功能, 且对PF公共寄存器只读, 各个模式的寄存器以256KB地址空间为最小粒度分开。
 - 支持操作系统将TM逻辑端口对、逻辑端口聚合及TRUNK按制式分组, 一个物理资源只能属于一种制式, 物理资源与VFID绑定关系可由PF半静态配置。
 - 支持操作系统将TM物理端口PORT按制式分组, 物理端口资源可共享给多个制式, 也可只分配给一个制式独享, 物理端口与VFID绑定关系可由PF半静态配置。
 - 支持基于模式的中断上报机制, 能区分VF的就按照VF上报中断, 否则就按PF上报中断。
 - 支持资源虚拟化, 即支持所接收的任务描述符中队列编号是一个VF内部从零开始的独立虚拟编号。



- 支持TM master口发送时，携带stream id以支持SMMU进行VA到PA的转换，每个VF对应一个stream id寄存器，由PF半静态配置。
- 支持每种制式的独立复位，VF通过对TM逻辑端口对的复位来实现制式复位，并且可根据具体场景选择在复位时是否释放Buffer给BMU。

说明

半静态配置是指在相关业务停止或未启动时，软件可对属性进行改配；静态配置是指软件必须在TM初始化时配置属性，初始化完成后不可改配；动态配置是指软件可在业务运行过程中改配属性。

须知

Hi309aTM将256个逻辑端口分为128个逻辑端口对，连续的两个逻辑端口构成1个逻辑端口对，PF基于逻辑端口对分配资源给VF。

若用户不按照文档对规格流程的限制进行配置，会产生无法预期的后果。

● 队列丢弃

- 支持基于所有正常转发队列进行RED/Tail丢弃，只基于队列缓存的包描述符个数实现此拥塞算法，支持两种模板号选择模式，由PF静态配置；第一种模式基于队列选择模板号，每个队列一组RED参数；第二种模式为DSCP模板模式，丢弃模板个数64个，每个RED模板内有64组RED参数，依据描述符内的DSCP域段进行参数选择，丢弃模板参数可由PF动态配置，丢弃使能基于队列可由VF动态配置。
- 支持色敏入队功能，对于标识为黄色的描述符在正常RED/Tail丢弃的基础上，进行二次Tail丢弃；二次Tail丢弃水线基于队列最大深度的1/8粒度进行配置，VF可基于队列配置二次丢弃水线。
- 支持基于发送至PPE/POE的发送端口进行反压超时丢弃处理，超时参数发送端口配置共用一个，超时后硬件上报中断，由VF配置端口丢弃使能，最终实现超时丢弃；配置端口丢弃后，入队处理丢弃，出队处理也丢弃。
- 支持指定逻辑端口丢弃，对应逻辑端口下队列入队处理丢弃，出队处理也丢弃，支持VF可动态配置。
- 支持指定队列丢弃，对应队列入队处理丢弃，出队处理也丢弃，支持VF可动态配置。
- 支持基于队列的拥塞状态上报和记录，软件可基于队列动态配置拥塞水线，TM通过上报中断方式上报队列拥塞状态，多个队列产生拥塞时，TM根据配置选择上报第一次拥塞信息或最后一次拥塞信息。
- 根据算法和配置需要丢弃的PD，TM将其通过配置接口推送给BMU进行Buffer释放。
- 支持基于队列的NO_BUFFER丢弃状态值的记录。

须知

NO_BUFFER丢弃属于异常丢弃，当RED/Tail丢弃水线配置不合理（丢弃水线大于队列缓存深度）时才会发生，配置合理时不会发生。

● 队列调度



- 支持L4（1024个）→L3（256个）→L2（64个）→L1（64个）四级调度。
 - L4→L3（实体队列→逻辑端口）：固定收敛比8:1或4:1，PF可半静态配置收敛比，调度算法SP+DRR，调度权重（weight）分为weight_b、weight_u两部分，生成的weight = weight_b << weight_u，VF动态配置weight参数，weight_b配置范围0~255，配置为0表示SP属性，weight_u配置范围0~20，多个节点SP调度时，队列偏移号越小优先级越高。
 - L3→L2（逻辑端口→逻辑端口聚合）：最大支持收敛比256:1，VF可半静态配置绑定关系，调度算法SP+DRR，调度权重（weight）分为weight_b、weight_u两部分，生成的weight = weight_b << weight_u，VF动态配置weight参数，weight_b配置范围0~255，配置为0表示SP属性，weight_u配置范围0~20，多个节点SP调度时，编号越小的节点优先级越高。

须知

L3级有C桶保证带宽机制，因此L3级调度优先级由高到底分别为：C桶SP+RR调度，P桶SP+DRR调度。

- L2→L1（逻辑端口聚合→Trunk端口）：最大支持收敛比64:1，VF可半静态配置绑定关系，调度算法SP+DRR，调度权重（weight）分为weight_b、weight_u两部分，生成的weight = weight_b << weight_u，VF动态配置weight参数，weight_b配置范围0~255，配置为0表示SP属性，weight_u配置范围0~20，多个节点SP调度时，编号越小的节点优先级越高。
- L1（Trunk端口）：调度算法RR调度。
- 支持逻辑端口优先级业务BYPASS功能，通过配置逻辑端口级调度SP属性实现。
- 支持VF动态改配逻辑端口到物理端口的绑定关系。
- 流量整形
 - L4（实体队列）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，流控模式半静态配置。
 - L3（逻辑端口）支持保证带宽双漏桶bps流控或单漏桶pps流控，流控参数动态可配置，流控模式半静态配置。
 - L2（逻辑端口聚合）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，流控模式半静态配置。
 - L1（Trunk端口）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，流控模式半静态配置。
 - bps流控参数：整形范围8kbps ~ 14.4Gbps，整形误差±16bps；令牌桶深为0MB~4MB。



须知

实际整形时由于包长都为Byte，所以整形参数单位也是以Byte为单位，1k/s=8kbit/s。

整形参数中的K表示 10^3 ，M表示 10^6 ，G表示 10^9 。

pps流控参数：整形范围4Kpps ~ 15Mpps，整形误差 ± 1 pps；令牌桶深为 16p~16Kp。

流控参数VF动态可配置，生效时间小于100 μ s。

- Port Trunk功能
 - 支持64个发送端口，其中发送端口0~25与PPE 26个物理端口一一对应，剩余发送端口推送POE，PF静态配置这些物理端口的发送地址。
 - Trunk选路功能在入TM前完成，即通过入TM描述符的TP[5:0]域段体现。
 - Trunk和对应的发送端口节点的绑定关系可配置，最大支持1: 8，即一个Trunk绑定8个发送端口。
 - 对于所有发送端口，一个发送端口可以最多被8个Trunk节点绑定，同时要求绑定在同一个发送端口的Trunk节点不得绑定其他的发送端口。
- 紧急队列
 - 支持64个发送端口每个对应一个紧急队列，每个紧急队列深度为8，独立于正常转发1024个队列，且发送优先级高于对应发送端口下的正常转发队列。
 - 紧急队列根据输入包任务里的Ur域段指示，直接由入口输出到出口，不进行入队和出队处理以及调度和整形处理。
 - 支持CPU通过POE添加紧急队列，支持加速器直接添加紧急队列。

须知

紧急队列出队只扣减trunk级令牌，但不受shaper限流。

当紧急队列的一个port被多个trunk绑定时，扣减trunk编号最大的shaper令牌；如果查不到其绑定的trunk，默认扣减trunk 63的shaper令牌。

- 中断上报
 - 支持内部使用FIFO空读满写中断。
 - 内部使用RAM，支持1bit ECC自动纠错并上报中断不回写、2bit ECC错误上报中断，并进行出错统计，所有1bit ECC计数器一个，2bit ECC计数器一个，对于TM内部所有memory，上报中断并记录错误地址。
 - 支持发送端口不在对应Trunk绑定关系中中断上报。
 - 支持发送端口反压超时中断上报。
 - 操作DDR总线返回非OK的中断上报。
 - TM进行回读操作且DDR返回OK时，对回读描述符的queue_id域进行合法性检查，即内部发起回读的queue_id和回读回来PD携带queue_id域段不一致，则上报中断并记录正确和错误的queue_id信息；若回读描述符的其他域段的错误，TM不会上报中断，并将此描述符转发给其他加速器。
 - 支持CPU读写寄存器行为异常中断。



- 支持分片添加顺序错误中断及分片个数超过最大个数限制错误中断。
 - 支持到达保序添加包SN号错误中断。
 - 支持添加描述符TM_TLV检查错误及各级节点绑定关系配置检查错误中断。
 - 支持所有中断都有对应的原始中断寄存器、中断屏蔽、中断注入寄存器。
 - 支持中断错误状态上报，并记录最早的错误中断信息，软件清除对应中断后，才能捕获新的中断信息。
 - TM内部能够区分VF上报的中断有：VF添加包TP_ID映射关系查询错误中断、VF添加包LOC_ID映射关系查询错误中断、BMU返回假满超过重传次数中断、VF配置寄存器映射关系查询错误中断、VF_FUNC_EN无效丢弃报文中断、添加报文绑定关系配置异常中断、添加报文TLV域段异常中断，其余中断均在PF上报。
- 业务统计功能
 - L3（逻辑端口）：支持逻辑端口出包的个数+字节数统计。
 - L4（实体队列）：支持实体队列出包的个数+字节数统计，丢弃包个数+字节数统计。
 - 基于Sid统计：支持基于1024个Sid的入口个数+字节数统计，每个描述符最多携带6个Sid，存储在PD对应的报文描述符内，描述符入队检查通过后会进行基于Sid的统计。
 - 如上统计为非饱和计数器，即计数满后自动溢出，不会保持最大值，且无中断上报，且存储方式为片内存储，包个数统计位宽48bit，字节数统计位宽64bit。

须知

CPU 读TM内部64bits位宽的统计值（包括队列缓存包字节数统计、队列发送包字节数统计、队列丢弃包字节数统计、逻辑端口缓存包字节数统计、逻辑端口发送包字节数统计、SID字节数统计）及48bits位宽的统计值（TM缓存的总字节数、队列发送包个数、队列入队包个数统计、队列丢弃包个数统计、队列缓存字节统计寄存器、SID的描述符个数统计）时，必须通过发出single 64读操作来实现，TM内部不会对数据进行额外锁存。

- 非业务统计功能
 - L4（实体队列）：支持实体队列入包个数，缓存包个数+字节数统计。
 - L3（逻辑端口）：支持逻辑端口缓存包个数+字节数统计。
 - L2（逻辑端口聚合）：支持逻辑端口聚合缓存包个数统计。
 - L1（Trunk端口）：支持Trunk端口缓存包个数统计。
 - CPU加包总个数统计，加速器加包总个数统计，正常转发队列和紧急队列各64个发送端口的个数统计。
 - 针对各加速器添加包描述符个数分别统计。
 - TM支持每类丢弃的分别统计，包括RED/Tail/色敏丢弃包个数统计，队列不使能入队丢弃包个数统计，发送端口不使能入队丢弃包个数统计，队列不使能出队丢弃包个数统计，发送端口不使能出队丢弃包个数统计，紧急队列丢弃包个数统计。
 - 如上统计为非饱和计数器，即计数满后自动溢出，不会保持最大值，且无中断上报，且存储方式为片内存储。



- DFX
- 支持Debug Bus接口可访问（读写）TM对用户开放的所有配置寄存器和表项，添加包描述符的寄存器空间不可访问。
- 支持安全复位功能。
- 支持模块内部状态机软件可见。
- 支持状态机智能心跳，当一个状态机长时间处在某个非IDLE状态时上报心跳停止中断，停留时间软件可配置。
- 支持FIFO空满标志软件可见。
- 支持内部MEMORY_ECC校验中断使能，可由PF配置。

须知

模块整体复位时，是逻辑出现挂死或其他不可恢复的问题，此复位不释放Buffer给BMU，且只复位TM AXI MASTER时钟域段的逻辑。

- 性能要求
出口性能单端口15Mpps。

须知

单端口即一个Trunk节点和一个发送端口绑定时的性能。

出口性能多端口20Mpps。

须知

多端口是指大于等于2个发送端口以上的性能。

- 时戳
使用Hi309a统一的时戳方案，使用描述符内的timestamp_tlv实现时戳功能，能够统计描述符在TM内的时延，时戳精度固定为0.1μs，宽度24bits。
PF静态配置TM时戳配置寄存器TM_TIME_STAMP_CFG，选择时间戳类型。

须知

当时戳类型为telemetry时，TM不看时戳配置，优先按照telemetry的要求进行时戳处理，详见telemetry章节。

- 时钟门控
TM支持时钟门控功能。当TM被配置为动态时钟门控使能有效，且内部没有任务转发时，硬件自动关断工作时钟来降低芯片的动态功耗。

2.4 遵循协议

AMBA4.0协议。

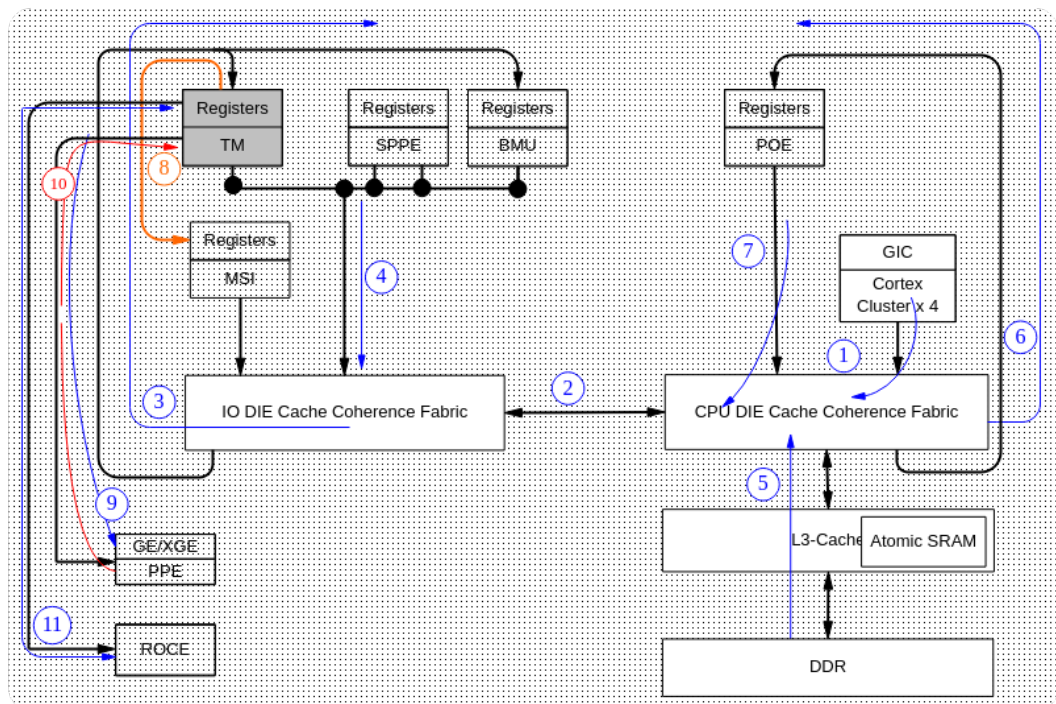
2.5 功能描述

2.5.1 应用框图

TM主要进行拥塞管理和流量整形，交给CPU进行处理的包流量或通过芯片接口发送的包流量，都可以通过TM进行流量整形和拥塞管理。

添加包任务到TM中的模块有CPU Core、POE、SPPE模块，在包描述符中会指定进入TM中的队列号。包任务经过调度和流量整形，TM会将包描述符推送给PPE、POE，且会将需要丢弃的包描述符里的Buffer地址等信息发送给BMU，由BMU完成后续Buffer资源回收。TM的应用如图2-1所示。

图 2-1 TM 应用图



应用图中：

- CPU Core通过①②③线路配置TM寄存器。
- CPU Core通过①⑥添加包任务到POE中，然后POE通过⑦②③将包推送到TM中。
- POE通过⑦②③包任务推送到TM。
- SPPE 通过④③将包任务推送到TM。
- TM通过线路⑨将包任务推送到PPE。



- PPE通过线路⑩硬反压TM。
- TM通过④②⑥线路将包任务推送到POE。
- TM通过④③线路将丢弃报文BUFFER释放给BMU。
- TM通过④②⑤线路对DDR进行读写。
- TM通过⑧线路上报中断。
- TM通过⑪线路和RoCEE模块交互，为RoCEE队列提供调度功能。

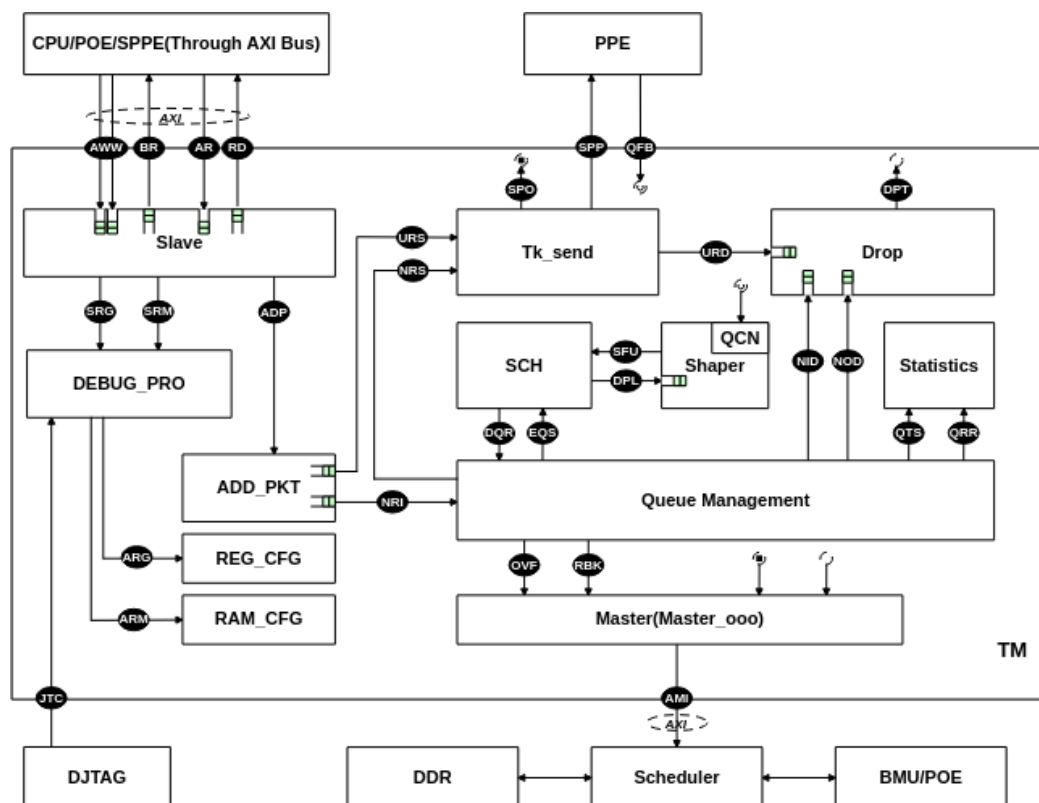
2.5.2 相关模块

在系统中，TM模块会和下面多个模块进行交互，分别描述如下：

- CPU Core：CPU Core通过POE将包任务添加到TM中，进行流量整形。
- POE：通过系统总线将包任务添加到TM中，进行流量整形；TM也可以将包任务进行流量整形后推送给POE。
- PPE：TM通过异步FIFO接口将整形后的包任务推送给PPE，PPE按照报文发送描述符对报文进行处理并通过网口发送出去。
- SPPE：可用于完成网络转发业务的端到端业务处理，包括报文解析、协议栈处理、转发决策、多播复制、报文修改封装以及业务处理过程中的CAR、统计等功能，SPPE会有两个添加TM的接口分别为EMIO和PKIO。
- BMU：TM将需要丢弃的描述符对应ADDR发送给BMU，由BMU完成Buffer释放动作。
- DDR：正常转发的包任务添加到TM的队列中，通过溢出回读到指定DDR空间中将其缓存。
- RoCEE：RoCEE内部缓存队列通知TM进行调度，TM将调度结果发送给RoCEE，RoCEE进行出队并更新队列信息。

2.5.3 模块框图

图 2-2 TM 模块框图



TM工作在两个时钟下，其中TM_SLV、TM_DEBUG_PRO、TM_RAM_CFG及TM_ADD_PKT的一部分工作在CLK_TM_S时钟域，频率为600MHz，如图2-2所示，其他所有子模块都工作在CLK_TM_M时钟域，频率为600MHz。两个时钟属于同频同源性时钟。

2.5.4 数据结构

TM读回的SID数据结构要求如图2-3所示。

图 2-3 TM 读回的 SID 数据结构要求

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
sv1	rsv					SID1										sv0	rsv					SID0									
sv3	rsv					SID3										sv2	rsv					SID2									
sv5	rsv					SID5										sv4	rsv					SID4									
RSV																															
说明																															
SV0~5		SID0~5 Valid																													
SID0~5		统计ID。																													



2.5.5 包描述符添加

- 在TM内，每个模式下有5组添加描述符接口，分别是POE_FWD_TM / POE_SEND_TM/EMIO_ADD_TM/PKIO_ADD_TM/RSV_ADD_TM，描述符位宽均为256bit。
- 所有加速器添加TM的Outstanding最大16。
- TM入口支持到达保序功能，对一组加包接口加入的PD，TM根据Hardware_TLV中的Otd_sn域段进行到达保序。
- MASTER添加包任务到TM，且队列片内片外总缓存个数已满时，TM会发生NO_BUFFER丢弃，将PD释放给BMU，并基于队列记录是否发生过NO_BUFFER丢弃的状态值（寄存器TM_FQ_NOBUFFER_DROP_STATE），一共1024bits，其中每个bit状态值可分别进行写1清操作。
- 当VF添加包描述符时，描述符中的tp_id和queue_id为虚拟ID值，需要硬件映射为物理ID进行处理，填写的tp_id和queue_id需要在软件配置的绑定关系范围内。
- PF不允许运行业务，当TM收到PF加包时，上报中断到PF，并记录伴随信息。
- 支持基于credit的添加机制，即TM_SLV收到包后，立即保序的返回Bresp；当到达保序完成后，返回credit值给对应MASTER，MASTER添加包时需要满足credit和outstanding均有剩余才能推送描述符给SLV。
- 默认所有MASTER的最大credit值均为最大outstanding的2倍。

须知

要求来自同一个报文的分片描述符的加包master、add_frag_id、队列号、port号必须相同，并且Add_frag_id在取值范围内，需要软件做好约束，否则TM会发生致命错误。

- TM收到首分片描述符后，会根据携带的pkt_num分片个数信息进行入队检查，做整体入队或丢弃处理和记录；当后续分片报文进入TM时，根据记录的信息进行入队或丢弃处理。
- TM硬件实际支持最大分片个数为255个，若实际添加分片个数大于pkt_num数据信息，则后续添加的分片描述符均做入队丢弃处理，TM上报中断。

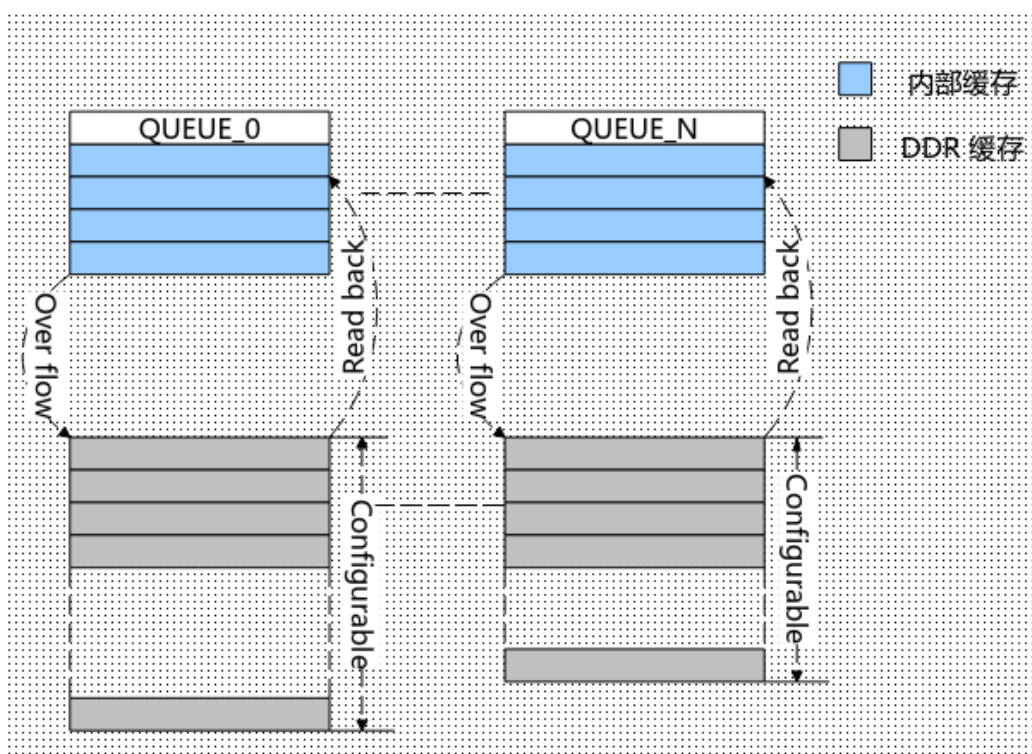
2.5.6 队列管理

- 采用片内存储方式+溢出DDR的机制，如图2-4所示，缓存包描述符。
- 正常转发1024每个队列都支持溢出回读，且溢出回读DDR基址和大小0~128*2047个描述符（单位128个，每个描述符占32Byte空间，则每个队列占DDR空间配置范围为0MB~8MB），由VF半静态且基于队列并按照页表粒度单独配置寄存器TM_QUEUE_DEPTH。

须知

- TM回读有低时延要求，建议将溢出回读行为配置为cacheble有效以降低时延。
- 队列深度取值决定了TM模块的抗突发能力，其典型值为4K个描述符，即建议配置为0x20。
- TM支持拥塞避免的丢弃机制，因此在应用场景上无法通过硬件保证不丢包的业务；如果应用中有相关场景，应通过将队列深度配置更大并且在加包之前查询TM的队列缓存个数统计来实现。

图 2-4 TM 队列管理机制



2.5.7 拥塞避免

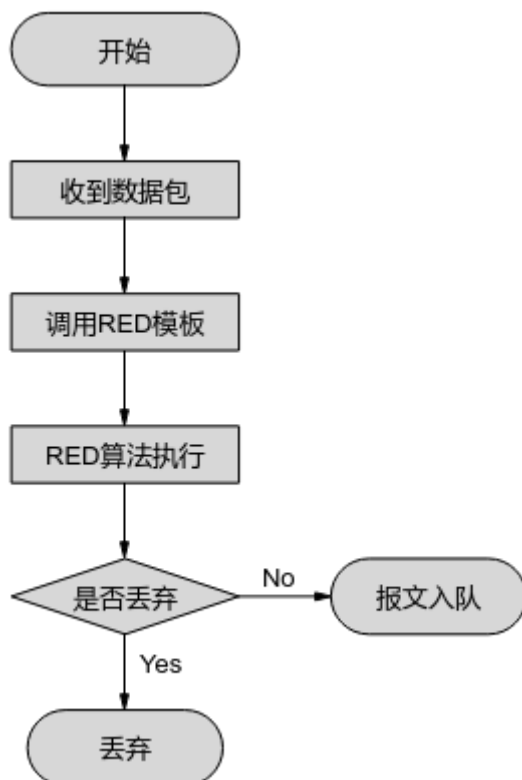
RED

RED丢弃算法，不是等缓存满后再丢弃到达的包，而是利用标记概率事先丢掉部分包来预防拥塞的发生。同时，RED 算法不是采用源抑制策略，立刻把反馈信息返回给发送端，而是通过设置标志位提示接收端，再由接收端传递给发送端；另外，RED 通过平均队列而非即时队列来调整分组丢弃概率，以期能尽可能地吸收部分短暂的突发流。可以避免TCP全局同步现象，用户可以设定队列的低门限和高门限。

- 当队列的长度小于低门限时，不丢弃包。
- 当队列的长度在低门限和高门限之间时，开始随机丢弃数据包，RED队列的长度越长，丢弃的概率越高。
- 当队列的长度大于高门限时，丢弃进行CHECK的数据包。

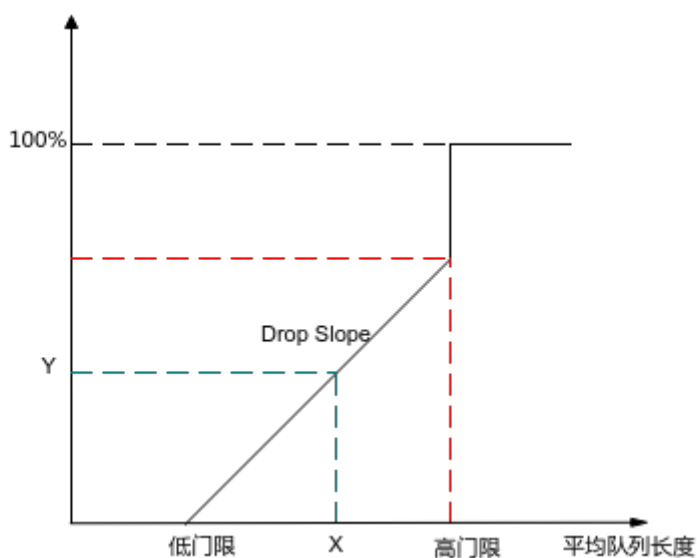
- 丢弃模板的参数有相关的表项存储。
- RED入队流程如图2-5所示。

图 2-5 RED 处理流程图



- RED丢弃曲线如图2-6所示。

图 2-6 RED 丢弃曲线示意图





- 队列当前长度为X，当低门限 $X \leq$ 高门限时，丢弃概率为Y， $0 < Y < 100\%$ 。
- 当 $X >$ 高门限时，包全部丢弃，丢弃概率 $Y = 100\%$ 。
- 当 $X \leq$ 低门限时，包全部通过，丢弃概率 $Y = 0$ 。

须知

Hi309a TM中基于标准RED算法做了简化处理，用瞬时队列长度代替平均队列长度进行RED算法计算，且计算出需要丢弃的包描述符则不入队，不会申请缓存空间，直接丢弃，即从队列的尾部丢弃而不是头部丢弃。

- RED算法在Hi309aTM中采用如下简化算法实现：
其中{min_thr, max_thr, drop_slope}参数，参见如表2-2所示。

表 2-2 RED 软件配置参数表

参数名称	描述
Max_thr	最大丢弃门限，单位为包个数，nManager中基于队列深度的百分比配置。
Min_thr	最小丢弃门限，单位为包个数，nManager中基于队列深度的百分比配置。
Drop_slope	Red丢弃斜率。

基于队列缓存包个数的算法：

```
temp0 = (current_queue_depth - min_thr)*drop_slope;
temp1 = (max_thr - min_thr)*random_num;
If (temp0 > temp1) {
Drop;
new_queue_depth = current_queue_depth;
}
Else{
Pass;
new_queue_depth = current_queue_depth + 1;
}
```

说明

- current_queue_depth：当前队列深度，单位包个数。
- random_num：随机数，所有队列共用。
- new_queue_depth：新队列深度，单位包个数。

Tail 丢弃

网络承载设备最原始的队列管理方法是通过对每个队列设置最大的队列长度，当队列没有满的时候，允许新的包添加，当队列满（或者到达某一门限）的时候，则拒绝或者丢弃新的包，直到队列又有空闲（或者到达某一门限）后，才接收新的包，这种技术就叫做Tail丢弃。

在TM中按照VF配置的队列流水线来实现。

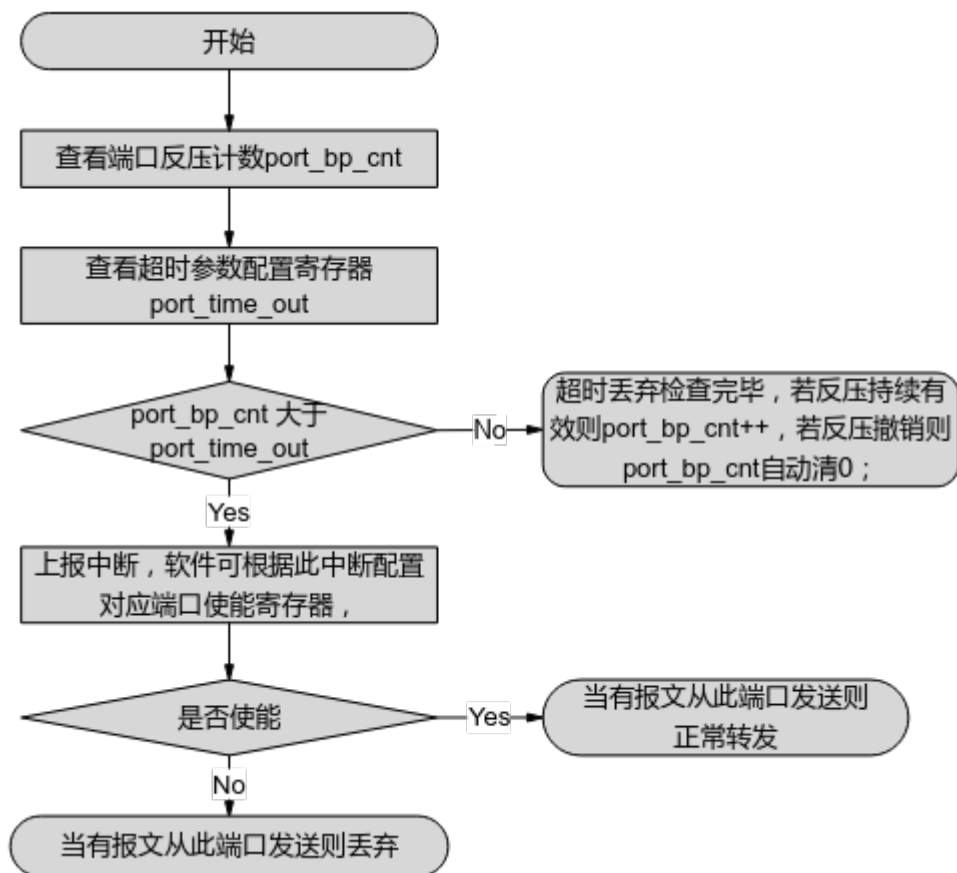
须知

TM对于缓冲区的拥塞避免采用RED和Tail丢弃两种方式，两种方式基于队列可根据用户配置任选其一，Tail丢弃模板和RED模板复用，且Tail丢弃复用RED丢弃最大丢弃门限参数，作为Tail丢弃门限。

超时丢弃

超时丢弃检测步骤和软硬件交互流程如图2-7所示。

图 2-7 TM 端口超时丢弃

**须知**

上述流程中软件对于超时的物理端口需要进行解绑定动作，即和对应的Trunk节点解除绑定关系，否则会造成对应Trunk节点下需要发送到其他物理端口的报文无法出包。

硬件内部port_bp_cnt计数逻辑：

```

If (bp_end || int_clr) {
    Port_bp_cnt = 0;
}
  
```

```

}
Else if (port_bp_cnt >= port_time_out){
Port_bp_cnt = Port_bp_cnt;
}
Else if (bp_vld){
Port_bp_cnt ++;
}
Else;

```

说明

- bp_end: 反压终止信号; int_clr: 中断清除。
- Port_time_out: 反压超时阈值; bp_vld: 反压有效。

指定逻辑端口丢弃

根据VF配置的逻辑端口丢弃使能表TM_LOC_EN，判断该逻辑端口下挂队列是否需要丢弃，若需要则对于入队的包描述符统一丢弃，且对于TM内部调度出的该逻辑端口下队列包描述符也统一丢弃，来完成此功能。

指定队列丢弃

根据VF配置的队列使能表TM_FQ_EN，判断该队列是否需要丢弃，若需要则对于入队的包描述符统一丢弃，且对于TM内部调度出的该队列的包描述符也统一丢弃，来完成此功能。

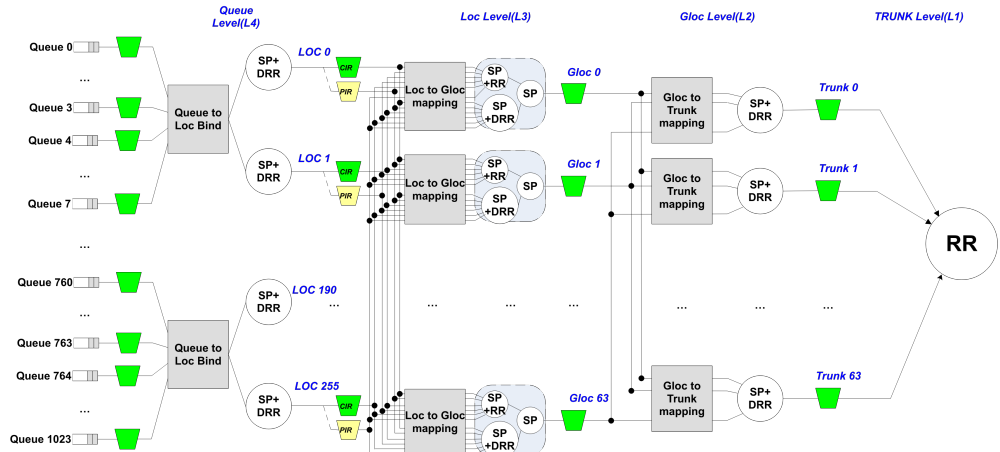
须知

当软件配置指定逻辑端口或队列不使能时，TM会启动“为了丢弃而出队”的行为，使不使能的逻辑端口和队列尽快排空。

2.5.8 队列调度

TM调度模式结构如图2-8所示。

图 2-8 TM 四级队列调度结构示意图





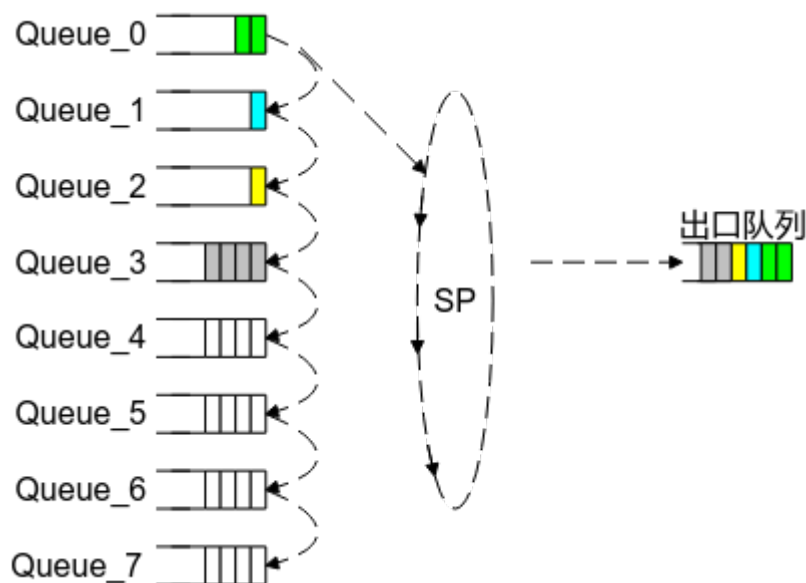
- L1调度，为虚拟队列Trunk端口节点，采用RR调度方式。
- L2调度，为虚拟队列逻辑端口聚合节点，采用DRR+SP调度方式，权重和SP/DRR属性用户可配；VF通过寄存器TM_GLOC_WEIGHT配置GLOC各节点权重值，当权重配置为0时，调度属性为SP，多个节点配置为SP调度属性时，编号越小的节点优先级越高。
- L3调度，为虚拟队列逻辑端口节点，逻辑端口使用双桶整形保证带宽机制，C桶相对于P桶固定SP，C桶之间进行SP+RR轮询，P桶之间进行SP+DRR调度方式，权重和DRR属性用户可配；VF通过寄存器TM_LOC_WEIGHT配置LOC各节点权重值，当权重配置为0时，调度属性为SP，多个节点配置为SP调度属性时，编号越小的节点优先级越高。
- L4调度，为实体队列流队列节点，采用SP+DRR调度方式，权重和SP/DRR属性用户可配；VF通过寄存器TM_FQ_WEIGHT配置Queue节点权重值，当权重配置为0时，调度属性为SP，多个节点配置为SP调度属性时，编号越小的节点优先级越高。
- 四级调度采用PULL的方式，即从L1到L4的方式调度出相应的实体队列；
- L4---->L3的链接关系固定8:1/4:1，基于逻辑端口可配置TM_FQ_BIND_CFG，且配置索引为L3_ID = L4_ID[9:3]。
- 查TM 队列绑定关系配置表（详见nManager）得到对应值：0b1，表示该L3_ID对应的两个逻辑端口都为4:1收敛比，即LOC_ID={L3_ID,0b0}和LOC_ID={L3_ID,0b1}；0b0，表示该L3_ID就是对应一个逻辑端口且为8:1收敛比，LOC_ID={L3_ID,0b0}。
- L3---->L2的链接关系可配置TM_LOC_LINK，最大255个L3属于同一个L2。
- L2---->L1的链接关系可配置TM_GLOC_LINK，最大64个L2属于同一个L1。
- 逻辑端口优先级业务BYPASS功能可以通过将高优先级L3（逻辑端口）下挂到对应L2（逻辑端口聚合），且该L2/L3的调度属性为SP。

须知

- 支持用户对队列调度结构的树形关系配置，但每个虚拟节点软件配置需要保证其唯一性，即一个L3（逻辑端口）只能属于一个L2（逻辑端口聚合节点），属于一个L1（Trunk）节点。
 - 当一个父节点下绑定的子节点属于多个制式时，此父节点必须属于PF配置，不可分配给任何VF配置；当父节点下绑定的子节点均属于同一制式时，此父节点可以分配给子节点的VF。
-
- Rocee队列的调度
 - TM支持对Rocee模块队列的调度，简称为Rschi调度。
 - Rsch不进行数据管理，只对Rocee队列进行调度，调度请求和结果均为硬连线接口。
 - Rsch基于quanta值调度，Roce模块比较队列的quanta水线和剩余quanta值向TM发起调度请求和响应，TM每次调度固定的quanta值发送给Roce模块用于队列出队，并使用quanta值更新对应队列的权重和令牌；quanta值由PF静态配置，范围256B~64KB。
 - Rsch的队列个数也由PF静态配置，范围0~128，当配置为N时，Rschi使用TM的0~（N-1）队列，TM内部使用N~1023。
 - 调度算法

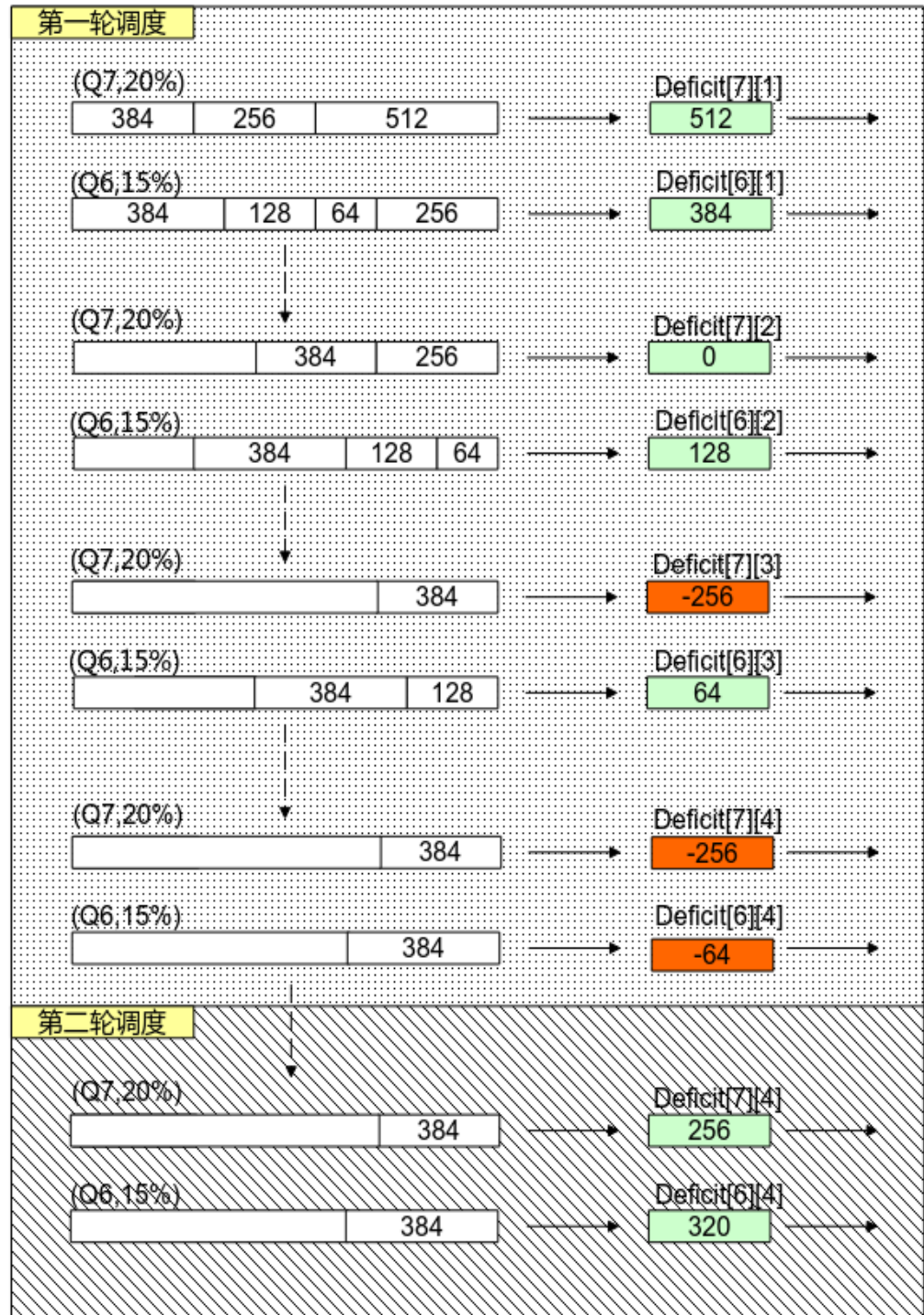
- RR (Round Robin) : 公平轮询各个队列，最简单的调度算法之一。
- SP (Strict Priority) : 基于优先队列的一种调度算法。该算法按照队列优先级的高低进行调度，高优先级先调度，低优先级后调度。高优先级队列非空，就先调度优先级高的队列出队，只有在优先级高的队列调度空后，才调度低优先级队列。队列优先级从高到低Queue_0> Queue_1> Queue_2> Queue_3> Queue_4> Queue_5> Queue_6> Queue_7，那么Queue_0有包时会一直调度此队列，直到队列空为止才开始调度队列Queue_1，依次类推，如图2-9所示。

图 2-9 SP 调度示意图



- DRR (Deficit Round Robin) 调度：同样也是RR的扩展，相对于WRR来言,解决了WRR只关心包个数，同等调度机会下大尺寸包获得的实际带宽要大于小尺寸包获得的带宽的问题，通过调度过程中考虑了包长的因素，从而达到调度的速率公平性。DRR 队列调度如图2-10所示。

图 2-10 DRR 调度示意图



假设用户配置各队列权重为4、3、2、1、4、3、2、1（依次对应Q7、Q6、Q5、Q4、Q3、Q2、Q1、Q0），权重对应偏移offset=7，则对应调度的Credit值为：512、384、256、128、512、384、256、128。如图2-10所示，队列Q7、Q6的能够分别获取20%、15%的带宽，当前Q7队列中有512Byte、256Byte、384Byte的包，Q6队列中有256Byte、64Byte、128Byte、256Byte的包。每次调度时，系统按权重为各队列分配带宽，假设Q7队列为400Byte/s，Q6队列为300Byte/s。Credit表示每次调度时各队列的带宽赤字。



- 第一次调度：
Credit[7][1]=128*4=512, Credit[6][1]=128*3=384, 从Q7队列取出512Byte包发送, 从Q6队列取出256Byte发送; 发送后, Credit[7][1]=0, Credit[6][1]=128。
- 第二次调度：
Credit[7][2]=0, Credit[6][2]=128, 由于Q7队列Credit 值=>0, Q7队列可以再被调度, 从Q7队列取出256Byte包发送, 从Q6队列取出64Bytes发送; 发送后, Credit[7][2]=-256; Credit[6][2]=64。
- 第三次调度：
Credit[7][3]=-256, Credit[6][2]=64, 由于Q7队列Credit值为负, Q7队列不会被调度; 从Q6队列取出128Byte发送; 发送后, Credit[6][2]=-64。
- 第四次调度：
系统设置Credit[7][4]=-256+512=256, Credit[6][4]=-64+384 =328, 由于Q7队列Credit值变为正, Q7队列再次被调度, 从Q7队列取出384Byte包发送, 从Q6队列取出384Byte发送; 发送后, Credit[7][1]=-128, Credit[6][1]=-64。如此这样循环调度, 最终Q7、Q6队列获取的带宽将分别占总带宽的20%、15%, 因此, 用户能够通过设置权重获取想要的带宽。

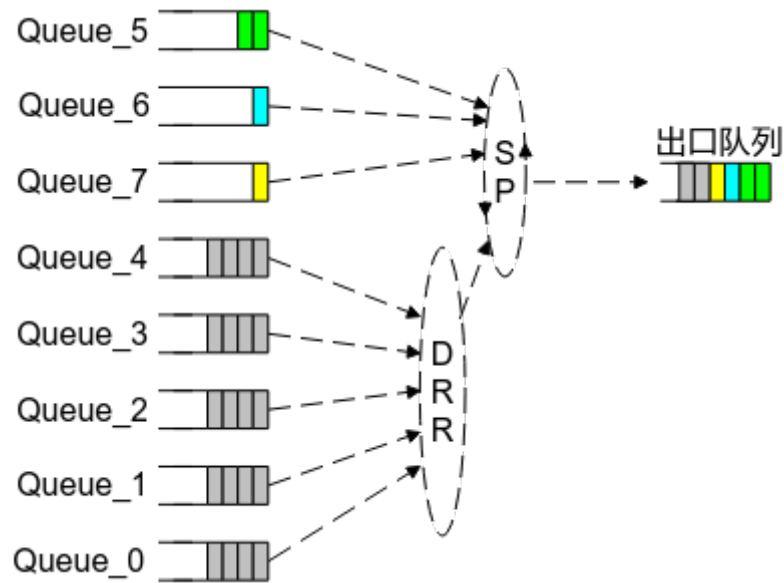
须知

DRR调度是基于字节的调度, 如weight比例为100:1, 最低的weight=1对应credit值设置为最大包大小 (常用1500Bytes), 则最高weight=100对应credit值为100*1500Byte。

Credit = weight_b(i) << weight_u(i), 如果配置credit和包长比例关系不合理, 例如最低的weight=1对应credit值设置为1, 则发送1500Bytes包后, 会进行根据如下算法进行credit刷新, 队列出队后若触发对应所有credit (i) 的刷新, 但刷新后所有credit(i)<0, 则后续触发刷新的条件则由空调度触发, 所以会造成出口带宽变小。

- SP+DRR调度: 用户可以配置队列的DRR参数, 根据配置将接口上的8 个队列分为两组, 一组 (例如Queue7、Queue6、Queue5) 采用SP 调度, 另一组 (例如Queue4、Queue3、Queue2、Queue1 和Queue0队列) 采用DRR 调度。调度示意图如图2-11所示。

图 2-11 SP+DRR 调度示意图



在调度时，首先按照SP方式调度Queue7、Queue6、Queue5队列中的包流，只有这些队列中的包流全部调度完毕后，才开始以DRR方式循环调度其他队列中的包流。Queue4、Queue3、Queue2、Queue1和Queue0队列包含自己的权值。重要的协议包和有低延时需求的业务包应放入采用SP调度的队列中，得到优先调度的机会，其余包放入以DRR方式调度的各队列中。

- 调度精度：

TM按照描述符中的PKT_LEN域段进行DRR调度，不会使用包长补偿值进行权重修正，因此在芯片出口采样，可能会出现调度误差增大的现象，且调度精度会随着不同DRR流之间PKT_LEN域段的差值而下降；

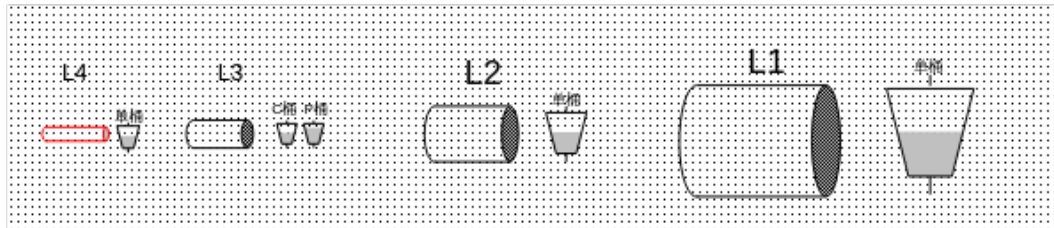
需要说明的是，调度精度的变化不会影响shaper的精度；

且PPE_SSU模块可以将补偿的前导码和帧间隙的值配置为0，这样可以消除包长补偿对DRR调度引入的误差。

2.5.9 流量整形

- L4（实体队列）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，寄存器TM_FQ_SHAPING。
- L3（逻辑端口）保证带宽双漏桶bps和单漏桶pps流控，整形属性半静态可配置，寄存器TM_LOC_SHAPING_TYPE，流控参数动态可配置，寄存器TM_LOC_CSHAPING、TM_LOC_PSHAPING。
- L2（逻辑端口聚合）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，寄存器TM_GLOC_SHAPING。
- L1（Trunk端口）单漏桶bps流控或单漏桶pps流控，流控参数动态可配置，寄存器TM_TK_PSHAPING。
- bps流控参数：整形范围8Kbps ~ 14.4Gbps，整形误差±16bps；令牌桶深为4KB~4MB。
- pps流控参数：整形范围4Kpps ~ 15Mpps，整形误差±1pps；令牌桶深为16p~16Kp。

图 2-12 TM 各层级整形方法示意图



- TM整形功能使用令牌桶来实现，即在可配置的填充周期内向令牌桶加入配置数目的令牌个数。当一个包被发送出去时，对应的令牌桶减掉相关的令牌个数。需要注意的是，当令牌桶已经被填充满时，额外的令牌将会被丢弃。
- 对于LOC级节点，其整形模式可以配置为pps或bps。如果配置为pps，相应的令牌桶每次减掉一个配置包长Pktsize（通过寄存器TM_SHAPING_PKT_LEN_CFG配置），则pps速率可以由下面公式进行计算：

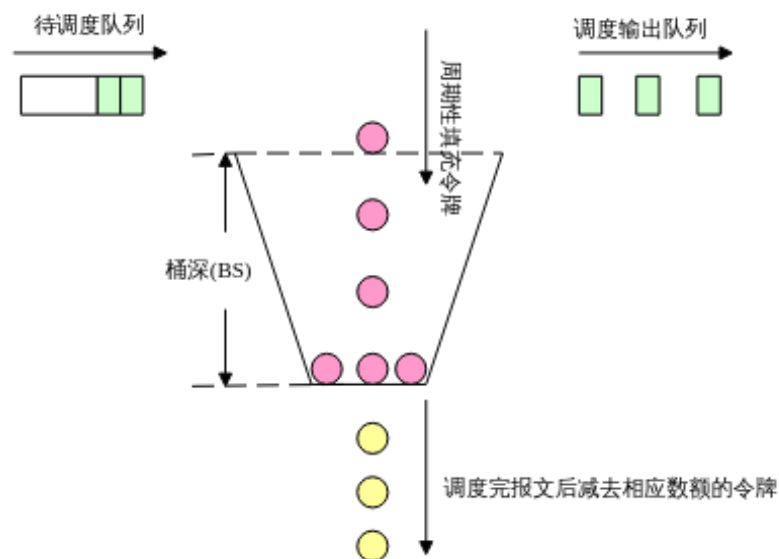
$$\text{PktRate (Mpps)} = \text{IR (Mbps)} / (8 * \text{Pktsize})$$
- C/P桶桶深计算公式为：

$$\text{xBS (Byte)} = \text{xBS_B} * 2^{\text{xBS_S}}$$

包长值补偿

- TM接收PPE的包长补偿值并对Shaper进行补偿。
- 补偿值信息中同样包含发送的包长和环回标志。
- Hi309a只针对1024个正常转发队列的各级shaper进行硬件补偿，紧急队列不会进行补偿，紧急队列报文可由软件或微码在填充Pkt_len域段时做软件补偿。
- 单桶整形原理

图 2-13 队列流量整形示意图

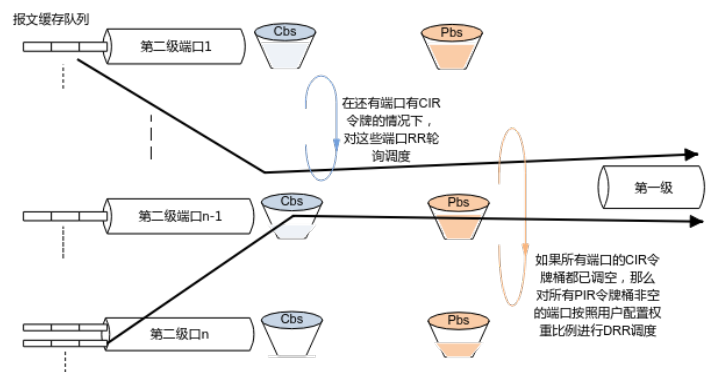


- 如图2-13所示，流量整形用于出口带宽的限制，包要被发送出去必须要有足够的令牌。作为服务质量功能的重要组成部分，一方面可以根据网络的实际规划为端口、用户、队列进行带宽分配，另一方面也可以起到流量平滑的作用。

须知

M里的整形方式采用令牌桶赤字方式，只要令牌桶里有令牌，就一定可以让此包通过，通过后可以将令牌桶赤字，并在后面令牌添加周期中补偿。

图 2-14 保证带宽流量整形示意图

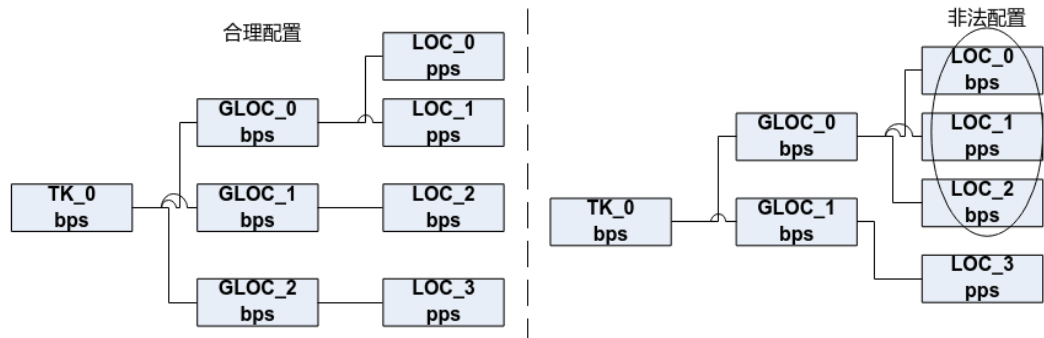


- 每个逻辑端口配置两级令牌桶，CIR/CBS和PIR/PBS令牌桶，并对逻辑端口的调度和整形算法进行修改。
- 优先以RR调度当前承诺带宽令牌桶（CBS）非空且有包要发的端口，并分别过CBS和PBS的令牌桶进行shaping处理，即通过C桶的包，CBS和PBS需要同时减去包对应的包长。
- 只有在所有包要发的端口的承诺带宽令牌桶（CBS）都已经为空的情况下，才按照用户配置权重，对（PBS）令牌桶中还有令牌的端口进行DRR调度，即通过P桶的包，只有PBS需要同时减去包对应的包长。

配置满足如下约束：

- 对同一逻辑端口：
 $L3Pir \geq L3Cir$, $L3Pbs \geq L3Cbs$, $L3Pbs \geq L3Pir$, $L3Cbs \geq L3Cir$ 。
 同一L2（逻辑端口聚合节点）下所有的L3（逻辑端口），需要软件保证配置L3Cir之和不能大于L2Cir。
- 整形参数配置规范：
 - 一个节点在正常使用时；承诺速率和桶深值均配置为非0值，要求令牌桶桶深足够大，满足至少一次添加动作不会使令牌桶由空变满。
 - 当关闭一个节点的流量时，需要将该节点的承诺速率和桶深值均配置为0；只配置其中一个参数为0的操作是不允许的。
 - 对于绑定在同一个父节点的所有子节点来说，子节点的整形类型配置必须保持一致，即统一配置为BPS整形或PPS整形；不允许在这些子节点中既有BPS类型的，又有PPS类型；配置约束如图2-15所示。

图 2-15 同级子节点整形类型配置约束



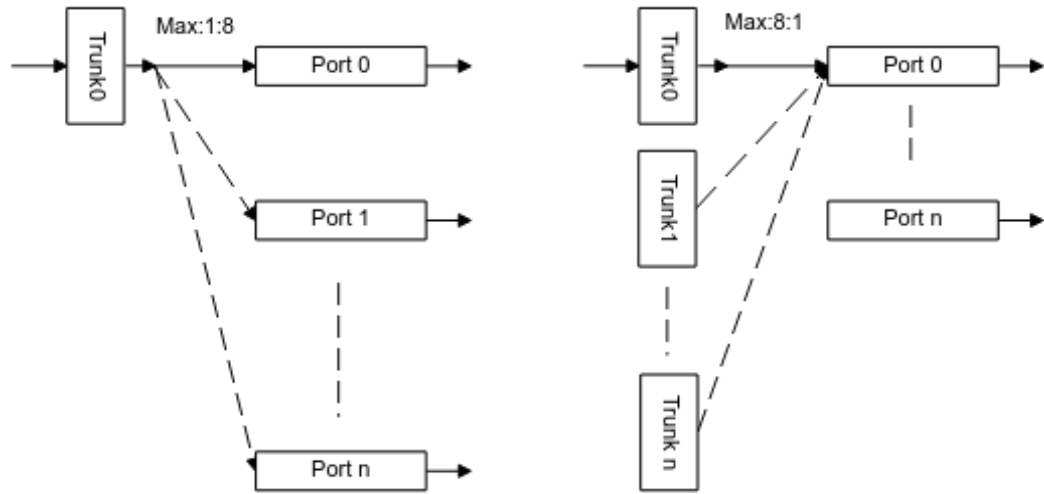
2.5.10 Port Trunk

TM支持Port Trunk功能，如图2-16所示一个Trunk端口可以绑定最多8个物理端口，绑定的物理端口范围为0~63；一个物理端口最多支持被8个Trunk端口绑定，且要求这些Trunk端口不能绑定其他的物理端口；从一个Trunk口调度出的描述符如，根据TP域段发送到指定的物理端口，Port Trunk绑定关系VF可半静态配置，寄存器TM_TK_BIND。

须知

当某个Port被反压时，与其有绑定关系的Trunk节点均会被反压。

图 2-16 Trunk 绑定物理端口示意图

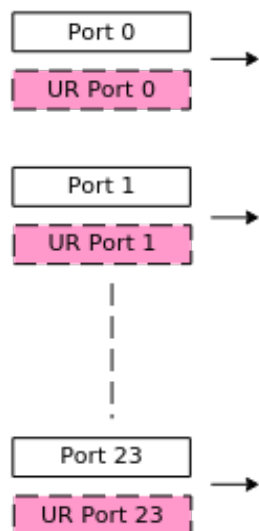


- 若TP不在对应Trunk的绑定范围内，TM上报中断；且该TP对应包任务需根据此Trunk的默认Default_TP发送出去，要求VF配置的Default_TP在此Trunk的绑定关系内；

2.5.11 紧急队列

- TM支持紧急队列功能，如图2-17所示每个物理端口都有一个对应的紧急队列和正常转发队列，是正常转发队列还是紧急队列由TM输入包描述符格式里的Ur域段决定，若是紧急队列则由TP决定对应的紧急队列端口。

图 2-17 物理端口对应紧急队列示意图



- 紧急队列应用场景主要转发一些协议报文，这些协议报文无需调度和整形，且需要低延时。
- 同一个物理端口下的紧急队列和正常转发队列之间为SP调度，紧急队列优先级高。
- TM每个物理端口紧急队列的推送Outstanding、推送类型、推送地址配置与普通队列共享。

须知

TM紧急队列的缓存固定为8深，当TM 物理端口拥塞时（例如物理端口长时间被反压），TM内部产生链路级反压传递到模块入口，出现HOL问题，导致TM总入口拥塞，影响所有队列的转发时延和丢包率。

2.5.12 反压处理

- TM接收端口（AXI SLAVE）
- TM 发送端口（AXI MASTER）
- Trunk调度节点，支持VF配置反压TM_TK_SOFT_BP，每个Trunk节点对应其中的软反压寄存器里的1bit；反压有效则停止该Trunk节点的调度，否则正常调度，此功能用于内部测试和调试。
- TM提供接受PPE硬反压实时状态寄存器（TM_PPE_BP）及反压次数统计寄存器（TM_PORT_TPU_BP_NUM_{0~25}），以便软件查询。
 - PPE硬反压共26bits，每bit反压信号固定绑定到TM的一个物理端口；反压有效后，对应节点停止调度；当反压长期有效并超过超时阈值，TM上报中断。



- Hi309aTM需要支持CPU后处理的场景，因此部分发送到POE的端口也需要响应PPE反压，TM支持发送到POE的端口配置对应的PPE反压，每个端口配置对应的反压线编号，反压有效后，对应节点停止调度。
- 支持POE反压；提供反压次数统计寄存器（TM_PORT_BP_NUM），以便软件查询。
 - 反压共6bits，每bit反压信号固定绑定到TM的一个发送端口；反压有效后，对应节点停止调度；当反压长期有效并超过超时阈值，TM上报中断。
 - 由于Hi309aTM支持拥塞丢弃报文捕获上送POE，该丢弃通道与推送POE业务报文共享添加接口和分配POE内部资源，因此反压可以配置反压到捕获接口上；反压有效后，丢弃捕获通路停止推送POE；当反压长期有效并超过超时阈值，TM上报中断。

2.5.13 多模隔离

详见《Hi309a MultiMode FS》，下面简述：

- Hi309a支持多种制式（RAT）同时运行，各个制式之间的业务处理需要隔离，采用基于虚拟化的机制来实现多模的隔离。
- TM通过PF-VF的方式进行功能及寄存器接口划分，共支持1个PF和5个VF；PF（Physical Function，物理功能）管理所有VF的公共功能以及各个VF之间的资源分配。VF（Virtual Function，虚拟功能）是能够独立完成业务处理的功能（如与物理端口、Trunk端口、逻辑端口聚合、逻辑端口、物理队列相关的配置寄存器等）。
 - PF和VF的寄存器支持按照页表粒度256KB隔离。
 - 通过配置PF寄存器可以完成VF所有资源的初始化、回收和动态/静态调整。
 - VF支持的软件可见的资源（例如TM的逻辑端口Loc资源）虚拟编号（V_Loc）与物理编号（P_Loc）解耦，即每个VF的资源虚拟编号都从0开始。
 - 对于PF中的每个VF的配置寄存器（如每个VF的映射表项），在各自VF中有对应的可读寄存器映射。
 - 所有VF具有偏移地址和含义都一致的公共可编程寄存器，如添加任务地址等。
 - 所有PF/VF中的软件任务下发接口支持多VCPU并发访问。

表 2-3 TM 寄存器划分多模列表

软件看到的VF/PF	模块寄存器地址接口	大小	起始地址	说明
PF	模块级	256KB	0x0~0x3FFF	模块头信息。
				模块级公共关键状态。
				模块级公共中断。
				模块级配置寄存器。
				模块级公共维测统计。



软件看到的VF/PF	模块寄存器地址接口	大小	起始地址	说明
	PF_VF0	256KB	0x40000~0x7FFFC	PF访问该VF的中断。
				PF访问该VF的业务配置。
				PF访问该VF的状态、业务维测。
				该业务资源做业务时的命令。
	PF_VF1	256KB	0x80000~0xBFFFC	PF访问该VF的中断。
				PF访问该VF的业务配置。
				PF访问该VF的状态、业务维测。
				该业务资源做业务时的命令。
	PF_VF2	256KB	0xC0000~0xFFFFC	PF作为VF处理业务时的中断。
				PF作为VF处理业务时的配置。
				PF作为VF处理业务时的状态、业务维测。
				PF作为VF处理业务时的命令。
	PF_VF3	256KB	0x100000~0x13FFFC	PF作为VF处理业务时的中断。
				PF作为VF处理业务时的配置。
				PF作为VF处理业务时的状态、业务维测。
				PF作为VF处理业务时的命令。
	PF_VF4	256KB	0x140000~0x17FFFC	PF作为VF处理业务时的中断。
				PF作为VF处理业务时的配置。
				PF作为VF处理业务时的状态、业务维测。
				PF作为VF处理业务时的命令。
VF0	业务级_VF0	256KB	PF最后2048K地址范围	芯片级寄存器（该VF只读）。
				该VF处理业务时的中断。
				该VF处理业务时的配置。
				该VF处理业务时的维测统计。
				该VF处理业务时的命令。
VF1	业务级_VF1	256KB	PF最后2048K地址范围	芯片级寄存器（该VF只读）。
				该VF处理业务时的中断。



软件看到的VF/PF	模块寄存器地址接口	大小	起始地址	说明
				该VF处理业务时的配置。
				该VF处理业务时的维测统计。
				该VF处理业务时的命令。
VF2	业务级_VF2	256KB	PF最后2048K地址范围	芯片级寄存器（该VF只读）。
				该VF处理业务时的中断。
				该VF处理业务时的配置。
				该VF处理业务时的维测统计。
				该VF处理业务时的命令。
VF3	业务级_VF3	256KB	PF最后2048K地址范围	芯片级寄存器（该VF只读）。
				该VF处理业务时的中断。
				该VF处理业务时的配置。
				该VF处理业务时的维测统计。
				该VF处理业务时的命令。
VF4	业务级_VF4	256KB	PF最后2048K地址范围	芯片级寄存器（该VF只读）。
				该VF处理业务时的中断。
				该VF处理业务时的配置。
				该VF处理业务时的维测统计。
				该VF处理业务时的命令。



图 2-18 虚拟逻辑端口对与物理逻辑端口对映射关系示意图

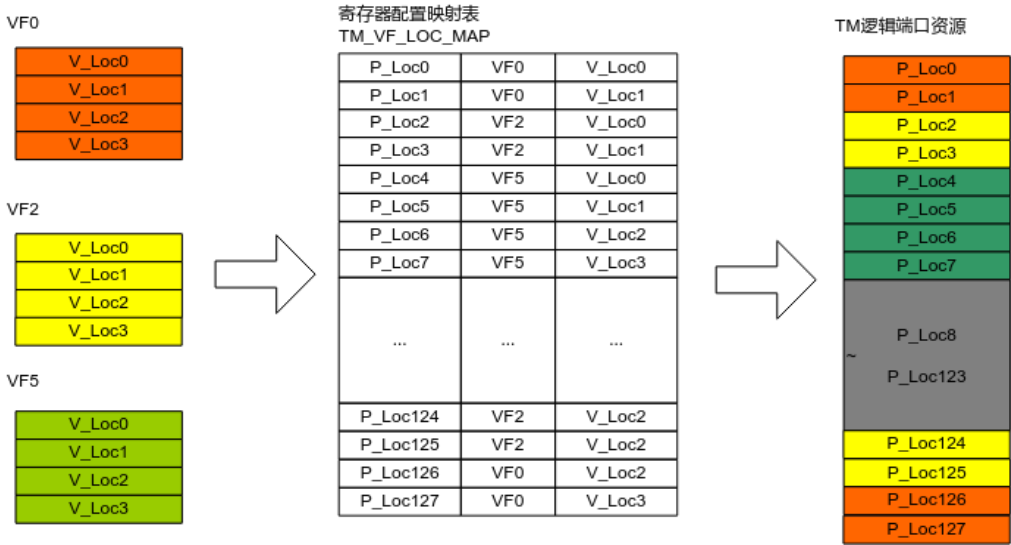


图 2-19 虚拟逻辑端口聚合与物理逻辑端口聚合映射关系示意图

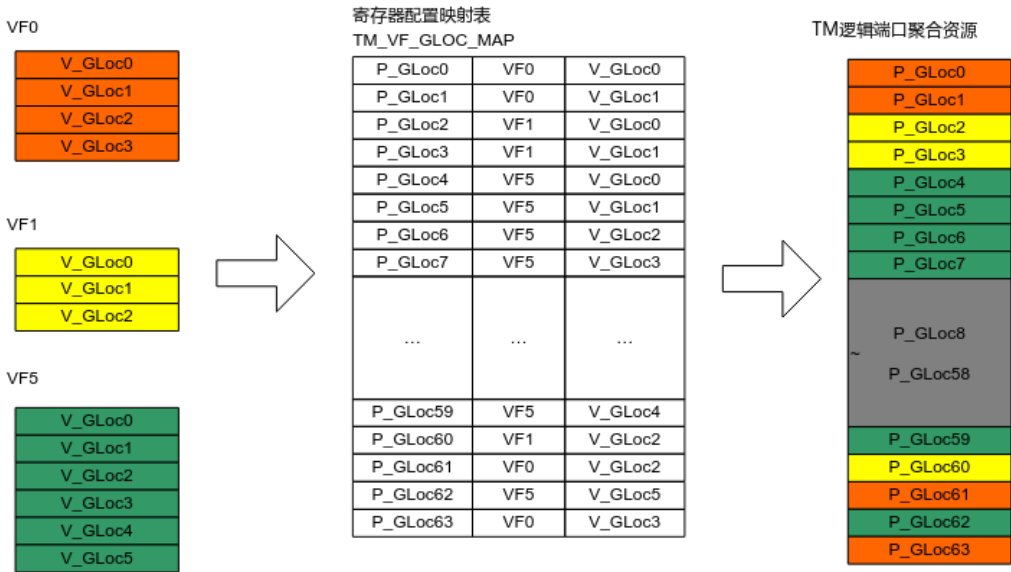


图 2-20 虚拟 Trunk 端口与物理 Trunk 端口映射关系示意图

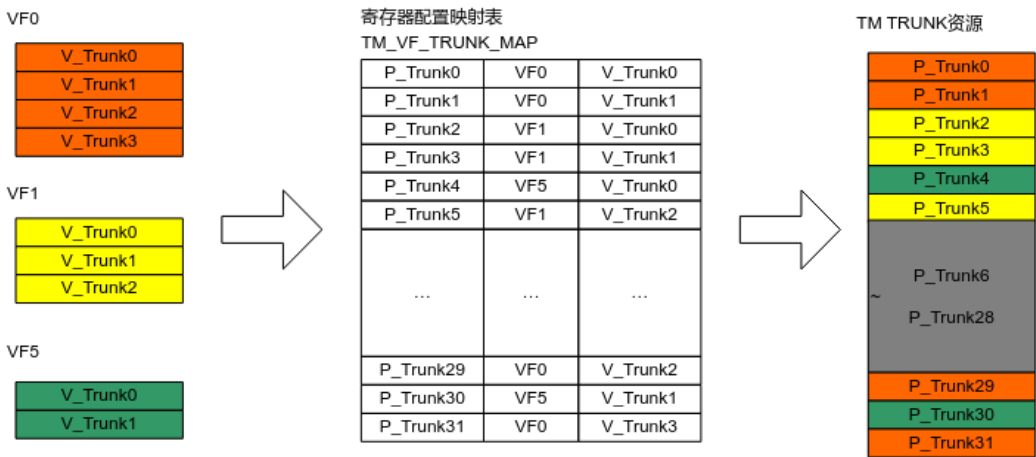
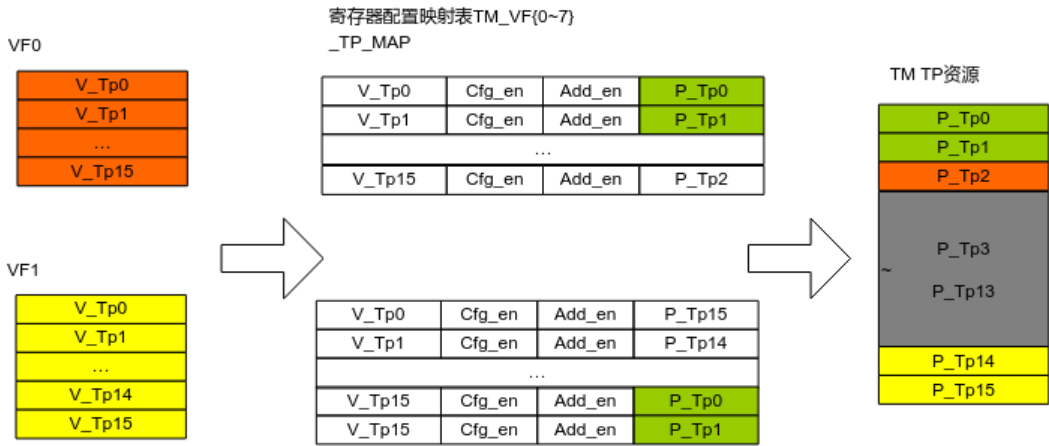


图 2-21 虚拟 TP 端口与物理 TP 端口映射关系示意图



- TM内部资源基于物理（TP）端口、Trunk端口、逻辑端口聚合、逻辑端口和PF/VF绑定。
- 对于Trunk、Gloc、Loc资源，每个节点只能属于一种制式，其资源分配原则是：所有资源默认都属于PF，若一个父节点（如Trunk）下挂的子节点（如Gloc）均为同一制式，则可以把这个父节点的权限也分配给该制式独享，否则，此父节点必须属于PF。
- 对于TP资源，一个TP可以被多个制式共享（因为MASTER推送TM时需要填入虚拟的TP_ID），但是当个TP被当个制式共享时，只能由PF来配置此TP的相关属性（如port_en），此时只能将对应映射关系的add_en域段置为有效，cfg_en域段置为无效；当个TP被某个制式独享时，此TP的配置权限可以对此制式开放，此时add_en和cfg_en域段均配置为有效。

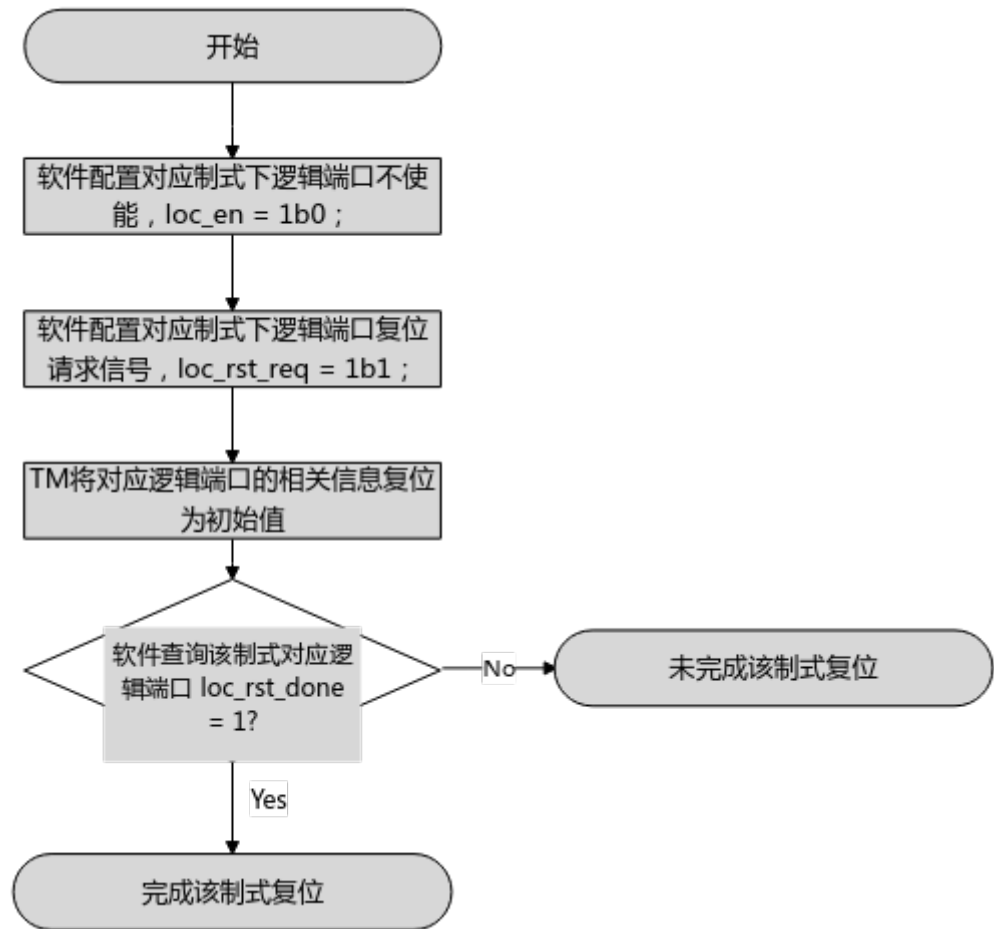


须知

对于Loc资源，其资源分配原则是：所有资源默认都属于PF，基于逻辑端口对来分配所属制式；TM将256个逻辑端口分为128个逻辑端口对，分配资源时，必须将某一个逻辑端口对整体分配给一个制式的虚拟逻辑端口对。

- 虚拟队列的配置：对于虚拟逻辑端口下的虚拟队列，TM并不提供映射表，而是复用逻辑端口的资源映射关系表来实现虚拟队列的映射。
 - 虚拟队列的映射过程。
 - 按照固定8：1的绑定关系找到虚拟队列所属的逻辑端口对。
 - 根据VF_ID和虚拟逻辑节点号查找物理逻辑端口对号。
 - 在物理逻辑端口对号中做与虚拟相同的偏移（0~7），即可得到虚拟队列号所对应的物理队列号。
- 每个PF/VF模式均有各自独立的添加包接口，支持多模并发添加TM。
- VF访问虚拟资源属性或添加包时，若该访问映射不到物理资源，TM会上报中断给该VF，记录相应的伴随信息，若为写操作则不生效，若为读操作则会返回错误的读数据。
- 支持基于PF/VF模式的推送地址选择，当TM作为MASTER访问其他模块时，会根据发送或读取数据的模式来选择对端模块的接口地址进行访问。
- 多模复位：每种制式支持两种复位模式：
 - 一种是将逻辑端口下挂的队列资源全部释放，由于TM的制式是和逻辑端口绑定，则复用逻辑端口不使能丢弃流程完成释放。
 - 另外一种复位模式，配置对应逻辑端口复位请求信号，逻辑完成内部复位操作，即将对应逻辑端口和队列相关值恢复为初始值，不释放Buffer给BMU。

图 2-22 TM 多模独立复位



2.5.14 中断上报

- 内部使用FIFO空读满写中断, 寄存器TM_FIFO1_RINT、TM_FIFO2_RINT、TM_FIFO3_RINT、TM_NEW_ADD_CHK_RINT、TM_URG_FIFO_RINT、TM_PORT_FIFO_RINT。
- 内部使用RAM的ECC单bit和2bit中断, 寄存器TM_RAM_CFG_ECC_ERR_RINT、TM_OTHER_RAM_ECC_ERR_RINT、TM_NEW_RAM_ECC_ERR_RINT。
- 总线异常操作中断上报, 主要是TM不支持的总线操作。
- TM不支持的总线添加, 如single 64bit ADD TM。
- 配置寄存器的非32bit总线配置操作, 如64bit配置操作。
- 物理端口不在对应Trunk绑定关系中中断上报, 寄存器TM_TK_BD_ERR_RINT。
- 物理端口和TRUNK节点反压超时中断上报, 寄存器TM_PORT_OUT_TIME_RINT、TM_TK_BP_TIME_OUT_ERR_RINT。
- TM在推送其他加速器时, 若总线返回Error, TM在寄存器配置的推送次数范围内仍无法推送成功的, 上报中断。
- TM内部状态机智能心跳中断上报, 寄存器TM_FSM_HBEAT_RINT。
- TM回读QID校验错误, 上报中断, 寄存器TM_NEW_ADD_CHK_RINT。
- 中断错误状态信息记录最早的错误信息。



2.5.15 统计功能

- L4 (实体队列) : 入包个数TM_FQ_INQ_NUM, 丢包个数TM_FQ_DROP_NUM + 字节数TM_FQ_DROP_BYTE0、TM_FQ_DROP_BYTE1, 出包个数TM_FQ_SD_NUM + 字节数TM_FQ_SD_BYTE0、TM_FQ_SD_BYTE1, 缓存包个数TM_FQ_ST_NUM + 字节数TM_FQ_ST_BYTE0、TM_FQ_ST_BYTE1, 片内统计。
- L3 (逻辑端口) : 出包的个数TM_LOC_SD_NUM + 字节数TM_LOC_SD_BYTE0、TM_LOC_SD_BYTE1, 缓存的包个数TM_LOC_ST_NUM + 字节数TM_LOC_ST_BYTE0、TM_LOC_ST_BYTE1, 片内统计。
- L2 (逻辑端口聚合) : 缓存的包个数TM_GLOC_ST_NUM, 片内统计。
- L1 (Trunk端口) : 缓存的包个数TM_TK_ST_NUM, 片内统计。
- CPU Core 加包总个数统计TM_CPU_ADDPKT_CNT, 加速器加包总个数统计TM_ACC_ADDPKT_CNT, 每类丢弃的分别统计, 64个发送端口的发送包个数统计TM_PORT_SEND_CNT。
- 对各加速器添加包描述符个数分别统计TM_ACC{0~4}_ADDPKT_CNT。
- 如上统计为非饱和计数器, 即计数满后自动溢出, 不会保持最大值。

2.5.16 DFX

- 支持Debug Bus接口可访问 (读写) TM对用户开放的所有寄存器和表项。
- 详安全复位流程。
- TM在初始化完成之后, 可以通过配置寄存器TM_CLK_GATE_EN为0x1, 使得TM模块在没有任务处理时整体关断时钟。
- 支持所有异常中断的中断注入。
- MEM ECC使能和错误植入, 支持1bit和2bit错误使能和植入。
- 现在的做法是根据寄存器配置对应MEM的ECC错误植入使能, 则会有对应的中断上报。

须知

实现只是触发中断, 并未经过真正的ECC逻辑, 但是可以保证中断通路的完整性。

2.5.17 时戳

- 对于加入TM包描述符任务, TM会记录描述符出入时间戳, 计算其在TM内部处理时延, 即 $\Delta T = T(\text{出}) - T(\text{入})$, 并根据时间戳配置, 将 ΔT 、 $T(\text{出})$ 、 $T(\text{入})$ 写入描述符的timestamp_tlv结构中, 也可配置为透传前级模块添加到TM的时间戳。
- $T(\text{出})$ 为TM将包推送给其他加速器的时刻, $T(\text{入})$ 为TM完整收到一个描述符的时刻。
- 时戳的粒度固定为0.1 μs 。
- 相关参数在寄存器TM_TIME_STAMP_CFG中由PF进行配置, TM时间戳值PF可读。



须知

当时戳类型为telemetry时，TM不看时戳配置，优先按照telemetry的要求进行时戳处理，请参见“[Telemetry捕获功能](#)”。

2.5.18 动态时钟门控

- 当TM内部逻辑处在空闲状态时，会自动关断工作时钟来降低芯片的动态功耗。
- TM动态时钟门控使能由PF静态配置，寄存器TM_CLK_GATE_EN。
- TM提供动态时钟门控DFX：门控时钟周期统计功能，寄存器TM_GATING_WINDOW、TM_ACC_GATING_STS。

须知

在业务数据进行转发过程中，软件若执行“[TM多模复位不释放Buffer步骤](#)”，TM动态时钟门控功能失效，即无法正常进入门控状态；除非软件对TM进行安全复位，才会使动态时钟门控功能恢复正常。

如果需要保证TM动态时钟门控功能在多模复位后仍然正确进行时钟门控，软件必须使用“[TM多模复位释放Buffer步骤](#)”对TM进行多模复位。

2.5.19 Telemetry 捕获功能

- 丢弃捕获：
 - 当timestamp_tlv为Telemetry时戳类型时，TM进行丢弃捕获，否则不进行丢弃捕获。
 - TM支持入口拥塞丢弃的捕获，具体包含入口的RED/TAI/色敏丢弃，其余丢弃类型不支持捕获功能。
 - 软件基于队列半静态配置捕获使能，使能有效时进行丢弃捕获，否则不进行丢弃捕获。
 - TM全局配置POE的VQI队列资源用于存放捕获报文，所有捕获报文推送给POE的指定VQI。
 - TM在出口将出口时戳值减去入队描述符内的时戳值的差值作为丢弃捕获timestamp。

须知

当捕获出口缓存拥塞后，后续捕获动作自动停止，认定为捕获失败，硬件分别统计捕获成功和失败的描述符个数。

- 超时捕获：
 - TM支持推送PPE的超长时延捕获，对于推送POE的描述符，不支持超时捕获功能。
 - 紧急队列不进行超时捕获。
 - 当timestamp_tlv为telemetry时戳类型时，TM进行超时捕获，否则不进行超时捕获；报文时延为TM出口时戳值减去入队描述符内的时戳值的差值。



- 当超时有效时，TM复制超时PD，原PD修改REL域段为不释放Buffer后继续推送给PPE，复制的PD推送给POE的指定VQI进行超时捕获。
- 软件基于端口半静态配置捕获使能，全局静态配置POE的VQI队列资源用于存放捕获报文，全局静态配置超时阈值。
- 数据结构：基于超时捕获的需求，TM推送PPE需要逐包增加QLEN[7:0]以及QPERCENT[5:0]，其中QLEN表示该描述符对应的队列缓存深度，具体值为： $QLEN[3:0] \times 2^{QLEN[7:4]}$ ，当缓存深度大于该公式表示的极限时，QLEN填全1；QPERCENT表示当前队列缓存占配置总缓存百分比，具体值为： $QPERCENT[5:0]/63 \times 100\%$ 。

须知

当捕获出口缓存拥塞后，后续捕获动作自动停止，认定为捕获失败，硬件基于端口进行捕获成功和失败的描述符个数统计。

2.6 软件指南

2.6.1 PF 初始化

须知

确保已经按照《Hi309a 通用MPU芯片 用户指南01-CPU小系统》的初始化顺序完成了初始化TM模块之前的其他初始化步骤。

表 2-4 TM 模块相关时钟复位配置寄存器

名称	寄存器地址	描述
SC_TM_ICG_EN	0xF5050340	TM时钟使能寄存器。
SC_TM_ICG_DIS	0xF5050344	TM时钟禁止寄存器。
SC_TM_RESET_REQ	0xF5050A50	TM模块软复位请求控制寄存器。
SC_TM_RESET_DREQ	0xF5050A54	TM模块软复位去请求控制寄存器。
SC_TM_ICG_ST	0xF5055340	TM时钟状态寄存器。
SC_TM_RESET_ST	0xF5055A50	TM模块软复位状态寄存器。

步骤1 配置表2-4寄存器SC_TM_ICG_EN值为0xF使能时钟、SC_TM_RESET_DREQ值为0x1F解复位TM。

步骤2 查询表2-4寄存器SC_TM_ICG_ST值为0xF、SC_TM_RESET_ST值为0x0，确保时钟有效且解复位成功，配置TM统计门控寄存器TM_GATING_WINDOW打开窗口。



- 步骤3** 配置TM_SLV_RAM_INIT_ENABLE为0x1，初始化内部配置类memory。
- 步骤4** 等待2μs以后，软件查询寄存器TM_SLV_RAM_INITIAL_DONE；若查询到寄存器TM_SLV_RAM_INITIAL_DONE为0x1，可开始执行软件初始化，否则继续**步骤6**，重复查询100次仍未完成则软件上报错误。
- 步骤5** 配置TM_RAM_INIT_ENABLE为0x1，初始化内部配置类memory。
- 步骤6** 等待2μs以后，软件查询寄存器TM_INITIAL_DONE；若查询到寄存器TM_INITIAL_DONE为0x1，可开始执行软件初始化，否则继续**步骤6**，重复查询100次仍未完成则软件上报错误。
- 步骤7** 配置TM_LGC_RAM_INIT_ENABLE为0x1，初始化内部逻辑的memory。
- 步骤8** 等待2μs以后，软件查询寄存器TM_LGC_INITIAL_DONE；若查询到寄存器TM_LGC_INITIAL_DONE为0x1，可开始执行软件初始化，否则继续**步骤8**，重复查询100次仍未完成则软件上报错误。
- 步骤9** 软件初始化配置TM模块，包括：TM工作模式（虚拟化映射关系表配置）、内部MEMORY属性配置、AXI总线接口设置、队列溢出回读基地址、队列深度、中断使能、DFX功能参数、返回CREDIT地址等需要静态配置的寄存器。

表 2-5 软件 PF 初始化需要配置的寄存器列表

名称	描述
DEVICE_ID	DEVICE_ID的设备ID。
SR_IOV_CTL	SR_IOV_CTL为VF使能寄存器。
SR_IOV_VF_OFFSET	SR_IOV_VF_OFFSET为每个VF相对于vf bar0的stride个数。
TM_PF_STREAM_ID	TM的PF的STREAM_ID配置寄存器。
TM_VF_STREAM_ID	TM的VF的STREAM_ID配置寄存器。
TM_PF_ID	TM的PF_ID址寄存器。
TM_VF_LOC_PAIR_MAP	逻辑端口对虚拟化映射关系表配置寄存器。
TM_VF_GLOC_MAP	逻辑端口聚合虚拟化映射关系表配置寄存器。
TM_VF_TRUNK_MAP	TM物理TRUNK与虚拟TRUNK的绑定关系配置寄存器。
TM_VF{0~7}_TP_MAP	TM VF{0~7}的虚拟TP编号与物理TP标号的绑定关系配置寄存器。
TM_RAM_CTRL_CFG	TM内部RAM配置寄存器。
TM_RAM_CFG_ECC_ENABLE	TM_RAM_CFG模块ECC错误使能寄存器。
TM_CNT_CLR_CE	TM CNT_CYC类型寄存器读清控制信号。
TM_TIME_STAMP_CFG	TM时戳功能相关配置寄存器。
TM_HBEAT_THLD	TM模块智能心跳阈值配置寄存器。



名称	描述
TM_SEND_BMU_SPADDR_VF	TM发送BMU VF分离buf配置地址。
TM_PORT_ADDR_VF	物理端口VF发送地址寄存器。
TM_SEND_POE/BMU_STRIDE_SIGN	TM推送POE/BMU时使用的SIGN。
TM_SEND_PPE/POE/BMU_VF_OFFSET	TM推送PPE/POE/BMU时使用的OFFSET。
TM_FQ_BIND_CFG	TM队列绑定关系配置寄存器。
TM_AXI_MST_CFG	TM MASTER AXI总线属性配置寄存器，由于总线延迟较大，需要软件在初始化时，将寄存器中的tm_send_awqos域段，配置为HIGH_QOS（0xB）。
TM_SEND_AWUSER_CFG{0,1}	TM推送加速器AWUSER配置。
TM_OVF_AWUSER_CFG{0,1}	TM溢出AWUSER配置。
TM_RBK_AWUSER_CFG{0,1}	TM回读ARUSER配置。
TM_DROP_AWUSER_CFG{0,1}	TM丢弃AWUSER配置。
TM_SHAPING_PKT_LEN_CFG	TM模块整形典型包长值配置。
TM_RED_TMPLT_SEL	RED模板的选择模式。
TM_RED_TMPLT_INFO	RED丢弃模板信息。
TM_CRDT_MST_ADDR	TM返回credit地址配置。
TM_SEND_PORT_HI_ADDR_PF/ TM_PORT_ADDR_PF	TM推送POE地址配置。
CFG_POE_MSG_BP_EN	POE的每个端口是否响应消息反压。
TM_OT_CAP_TIME_CFG	TM超时捕获超时时间配置寄存器。
TM_OT_CAP_EN_CFG	TM超时捕获使能。
TM_CAP_VQI	TM捕获报文的VQI值。
TM_FIFO_AFUL_GAP_CFG0/ CAP_FIFO_AFUL_GAP	TM捕获FIFO的水线，配置为5'h1F。

步骤10 软件初始化完成后，配置寄存器TM_CFG_OK为0x1，则可以正常处理包任务。

步骤11 解复位TM使用的PPE软复位信号，通过配置TM所属子系统下的系统控制器寄存器实现。

----结束



2.6.2 TM VF 初始化

- 步骤1** 软件查询寄存器TM_CFG_OK，确认TM PF是否完成初始化；若查询到寄存器TM_CFG_OK为0x1，可继续**步骤2**，否则重复**步骤1**，查询100次仍未完成则上报错误。
- 步骤2** VF初始化配置TM模块，包括：VF内部RED配置（RED使能）、VF内部节点使能（物理端口使能、逻辑端口使能、队列使能）、VF内部各级节点组网关系（各级节点间的绑定关系、各级调度整形参数配置）、队列溢出回读基地址、队列深度、队列信息表等需要半静态配置的寄存器。

表 2-6 软件 VF 初始化需要配置的寄存器列表

名称	描述
TM_FQ_EN	队列使能寄存器。
TM_LOC_EN	逻辑端口使能寄存器。
TM_PORT_EN	物理端口使能寄存器。
TM_PORT_DEF	默认物理端口寄存器。
TM_TK_BIND	TRUNK绑定物理端口表。
TM_TK_SHAPING_IR	TRUNK级整形参数。
TM_TK_SHAPING_BS	TRUNK级整形参数。
TM_TK_SHAPING_TYPE	TRUNK级整形类型。
TM_GLOC_LINK	逻辑端口聚合节点和Trunk聚合节点链接表。
TM_GLOC_WEIGHT	逻辑端口聚合权重表。
TM_GLOC_SHAPING_IR	逻辑端口聚合整形参数。
TM_GLOC_SHAPING_BS	逻辑端口聚合整形参数。
TM_GLOC_SHAPING_TYPE	GLOC级整形类型。
TM_LOC_WEIGHT	逻辑端口权重表。
TM_LOC_SHAPING_TYPE	逻辑端口整形类型。
TM_LOC_CSHAPING_IR	逻辑端口C桶整形参数。
TM_LOC_CSHAPING_BS	逻辑端口C桶整形参数。
TM_LOC_PSHAPING_IR	逻辑端口P桶整形参数。
TM_LOC_PSHAPING_BS	逻辑端口P桶整形参数。
TM_OVF_BASE_ADDR_[LOW,HIGH]	队列溢出回读基地址配置。
TM_FQ_SHAPING_IR	队列级SHAPING IR参数。
TM_FQ_SHAPING_BS	队列级SHAPING BS参数。



名称	描述
TM_FQ_SHAPING_TYPE	队列级SHAPING类型。
TM_FQ_RED_TMPLT_ID	TM队列RED信息表。
TM_LOC_LINK	逻辑端口节点和逻辑端口聚合节点链接表。
TM_QUEUE_DEPTH	队列深度。
TM_FQ_WEIGHT	队列权重表。
TM_YELLOW_PKT_CFG	TM黄色描述符配置寄存器。

步骤3 VF配置TM_VF_FUNC_EN为0x1，初始化完成。

----结束

2.6.3 TM 节点链接关系动态配置

TM组网信息中的绑定关系可以有条件动态配置。若需要将某调度节点的绑定关系剥离或将该节点添加到其他节点组网信息中，以L3（逻辑端口）节点为例，应遵循以下步骤。

- 步骤1** 读出LOC节点的Shaper配置TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS并记录，然后将LOC节点的Shaper配置TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS均改配为0x0。
- 步骤2** 完成**步骤1**后，等待800μs，然后配置TM逻辑端口使能寄存器TM_LOC_EN，使需要排空的L3（逻辑端口）节点不使能，排空其下队列缓存的包描述符，是否排空需要查询对应节点的统计值TM_LOC_ST_NUM是否为0；若为0，进入**步骤4**；若不为0，则进入**步骤3**。
- 步骤3** 扫描寄存器TM_ADD_FRAG_CNT{0…151}，若查询到域段tm_add_frag_cnt均为0，则软件上报告警；若查询到域段tm_add_dec_qid等于需要排空的Queue_id且域段tm_add_frag_cnt不为0，则软件根据对应的add_frag_id（寄存器地址索引）/Queue_id构造210个分片描述符添加TM（不需要申请Buffer，rls_ind填为不需要释放Buffer即可），添加完成后查询TM_LOC_ST_NUM是否为0，若重复查询1万次不为0软件上报告警，若为0则继续**步骤4**。
- 步骤4** 确定排空后，清除对应L3（逻辑端口）和L2（逻辑端口聚合）的绑定关系：软件读出原有绑定关系配置TM_LOC_LINK，然后将其中的配置使能域段cfg_en改为0，gloc_id保持不变，写回TM。

须知

如果需要排空的逻辑端口是ROCEE队列的，则需要通过排空ROCEE请求或者对ROCEE主动进行制式复位来实现该逻辑端口的排空，软件通过查询状态寄存器TM_ROCE_QUEUE_EGBMP{0…3}判断队列是否排空。
该警告对于动态改配L3到L2，L2到L1，L1到PORT的绑定关系流程均有效。



步骤5 重新配置L3（逻辑端口）和L2（逻辑端口聚合）的链接关系TM_LOC_LINK。

步骤6 将**步骤1**记录的Shaper配置TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/
TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS重新配置到对应LOC节点。

步骤7 配置完毕后，可以重新使用此节点。

----结束

再以L2（逻辑端口聚合）节点为例，应遵循以下步骤。

步骤1 读出GLOC节点的Shaper配置TM_GLOC_SHAPING_IR/TM_GLOC_SHAPING_BS并记录，然后将GLOC节点的Shaper配置TM_GLOC_SHAPING_IR/TM_GLOC_SHAPING_BS改配为0x0。

步骤2 配置TM逻辑端口聚合节点下所有逻辑端口或队列使能寄存器TM_LOC_EN/
TM_FQ_EN，使需要排空的L2（逻辑端口聚合）节点不使能，排空其下队列缓存的包描述符，是否排空需要查询对应节点的统计值TM_GLOC_ST_NUM是否为0；若为0，进入**步骤4**；若不为0，则进入**步骤3**。

步骤3 扫描寄存器TM_ADD_FRAG_CNT{0...151}，若查询到域段tm_add_frag_cnt均为0，则软件上报告警；若查询到域段tm_add_dec_qid等于需要排空的Queue_id且域段tm_add_frag_cnt不为0，则软件根据对应的add_frag_id（寄存器地址索引）/Queue_id构造210个分片描述符添加TM（不需要申请Buffer，rls_ind填为不需要释放Buffer即可），添加完成后查询TM_GLOC_ST_NUM是否为0，若重复查询1万次不为0软件上报告警，若为0则继续**步骤4**。

步骤4 确定排空后，清除对应L2（逻辑端口聚合）和L1（TRUNK）的绑定关系：软件读出原有绑定关系配置TM_GLOC_LINK，然后将其中的配置使能域段cfg_en改为0，trunk_id保持不变，写回TM。

步骤5 重新配置L2（逻辑端口聚合）和L1（TRUNK）的链接关系TM_GLOC_LINK。

步骤6 将**步骤1**记录的Shaper配置TM_GLOC_SHAPING_IR/TM_GLOC_SHAPING_BS重新配置到对应GLOC节点。

步骤7 配置完毕后，可以重新使用此节点。

----结束

再以L1（TRUNK）到物理端口的绑定关系改配为例，应遵循以下步骤。

步骤1 读出TRUNK节点的Shaper配置TM_TK_SHAPING_IR/TM_TK_SHAPING_BS并记录，然后将TRUNK节点的Shaper配置TM_TK_SHAPING_IR/TM_TK_SHAPING_BS改配为0x0。

步骤2 配置TM TRUNK节点下所有逻辑端口或队列使能寄存器TM_LOC_EN/ TM_FQ_EN，使需要排空的L1（TRUNK）节点不使能，排空其下队列缓存的包描述符，是否排空需要查询对应节点的统计值TM_TK_ST_NUM是否为0；若为0，进入**步骤4**；若不为0，则进入**步骤3**。

步骤3 扫描寄存器TM_ADD_FRAG_CNT{0...151}，若查询到域段tm_add_frag_cnt均为0，则软件上报告警；若查询到域段tm_add_dec_qid等于需要排空的Queue_id且域段tm_add_frag_cnt不为0，则软件根据对应的add_frag_id（寄存器地址索引）/Queue_id构造210个分片描述符添加TM（不需要申请Buffer，rls_ind填为不需要释放Buffer即可），添加完成后查询TM_TRUNK_ST_NUM是否为0，若重复查询1万次不为0软件上报告警，若为0则继续**步骤4**。



步骤4 确定排空后，直接重新配置L1（TRUNK）到物理端口的链接关系TM_TRUNK_BIND。

步骤5 将**步骤1**记录的Shaper配置TM_TK_SHAPING_IR/TM_TK_SHAPING_BS重新配置到对应TRUNK节点。

步骤6 配置完毕后，可以重新使用此节点。

----结束

2.6.4 TM 节点 Shaping 参数动态配置

步骤1 初始化时每个Shaping参数TM_TK_SHAPING_IR/TM_TK_SHAPING_BS/TM_GLOC_SHAPING_IR/ TM_GLOC_SHAPING_BS/ TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS/TM_FQ_SHAPING_IR/TM_FQ_SHAPING_BS都有对应的值。

步骤2 计算目的速率对应的各个参数值，然后将参数值直接配置到TM的对应寄存器。

----结束

2.6.5 TM 端口超时丢弃步骤

步骤1 软件检测到逻辑上报的物理端口反压超时中断TM_PORT_OUT_TIME_RINT。

步骤2 软件配置该物理端口不使能TM_PORT_EN，阻止TM入口报文对该端口对应队列的入队（如果需要切换端口可以修改描述符里的TP域段，软件入TM时做此动作）。

步骤3 将此物理端口从对应Trunk节点解除绑定TM_TK_BIND。

步骤4 逻辑将该物理端口下的报文描述符全部清空，即释放地址给BMU。

步骤5 软件读取物理端口下的队列缓存包个数TM_FQ_ST_NUM是否为0，确认是否清空，如果没有清空，继续**步骤5**，如果重复查询1万次没有清空，则软件上报告警。

步骤6 软件处理反压超时的原因。

步骤7 重新将此物理端口绑定到相关的Trunk节点TM_TK_BIND。

步骤8 软件配置物理端口使能TM_PORT_EN，恢复正常工作。

----结束

2.6.6 TM 多模复位不释放 Buffer 步骤

步骤1 软件配置需要复位制式下的逻辑端口使能寄存器TM_LOC_EN为不使能，即loc_en = 0b0。

步骤2 软件配置需要复位制式下的逻辑端口复位请求寄存器TM_LOC_RST_REQ有效，即loc_rst_req = 0b1。

步骤3 软件查询需要复位制式下的逻辑端口复位完成寄存器TM_LOC_RST_DONE是否有效，即loc_rst_done = 0b1。

步骤4 若软件查询需要复位制式下的逻辑端口复位完成信号全部有效，并等待1ms后，则完成该制式复位，需要再使用时将复位loc_rst_req=0b0，如果重复查询1万次复位完成信号均为无效，则软件上报告警。

----结束



2.6.7 TM 多模复位释放 Buffer 步骤

步骤1 读出LOC节点的Shaper配置TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS并记录，然后将LOC节点的Shaper配置TM_LOC_CSHAPING_IR/TM_LOC_CSHAPING_BS/TM_LOC_PSHAPING_IR/TM_LOC_PSHAPING_BS均改配为0x0。

须知

- 软件在多模复位时，需要保证不能持续向TM添加任务，否则无法复位。
- 保证不能持续向TM添加任务，否则无法复位。

步骤2 完成**步骤1**后，等待800μs，软件配置需要复位制式下的逻辑端口使能寄存器TM_LOC_EN为不使能，即loc_en = 0b0，排空其下队列缓存的包描述符，是否排空需要查询对应节点的统计值TM_LOC_ST_NUM是否为0；若为0，进入**步骤4**；若不为0，则进入**步骤3**。

步骤3 扫描寄存器TM_ADD_FRAG_CNT{0…151}，若查询到域段tm_add_frag_cnt均为0，则软件上报告警；若查询到域段tm_add_dec_qid等于需要排空的Queue_id且域段tm_add_frag_cnt不为0，则软件根据对应的add_frag_id（寄存器地址索引）/Queue_id构造210个分片描述符添加TM（不需要申请Buffer，rls_ind填为不需要释放Buffer即可），添加完成后查询TM_LOC_ST_NUM是否为0，若重复查询1万次不为0软件上报告警，若为0则继续**步骤4**。

步骤4 软件配置需要复位制式下的逻辑端口复位请求寄存器TM_LOC_RST_REQ有效，即loc_rst_req = 0b1。

步骤5 软件查询需要复位制式下的逻辑端口复位完成寄存器TM_LOC_RST_DONE是否有效，即loc_rst_done = 0b1。

步骤6 软件查询需要复位制式下的逻辑端口复位完成信号全部有效，并等待1ms后，则完成该制式复位，需要再使用时将复位loc_rst_req=0b0。

----结束

2.6.8 TM 安全复位不释放 Buffer 步骤

步骤1 将POE推送TM的地址动态改配为一个加包的reserve的地址，此时TM不会返回credit，各MST不会持续泄露Buffer，查询到各MST的credit耗尽时MST即停流。

步骤2 SPPE停止入口流量，并保证所有添加到TM的任务都已成功（通路处于空闲状态）。

步骤3 软件配置寄存器TM_MODULE_RST_REQ为0x1。

须知

软件在安全复位时，需要保证停止向TM添加任务，否则无法复位。

步骤4 软件查询寄存器TM_MODULE_RST_DONE；若查询到寄存器TM_MODULE_RST_DONE为0x1，则继续**步骤4**，否则重复**步骤3**。



- 步骤5** 配置TM所属SUBSYS系统控制器的PPE软复位有效，若TM有部分队列分配给ROCEE（通过TM_ROCE_QUEUE_NUM_CFG配置），则需要软件配置TM所属SUBSYS系统控制器的ROCEE软复位有效。

须知

软件在进行复位时，需要保证ROCEE和TM的复位有效时间有重叠部分，即在解复位任意一个模块之前，需要保证两个模块都已经复位有效。

建议的复位流程是：复位TMà复位ROCEEà解复位ROCEEà解复位TM。

- 步骤6** 配置寄存器TM_SLV_ACC_WR_PTR_CLR，清除TM_SLV到达保序对应的bitmap和写指针。
- 步骤7** 配置寄存器TM_TK_BIND值为全0，清除表项配置。
- 步骤8** 按照系统复位流程，配置TM所属SUBSYS系统控制器的TM软复位有效，完成复位后进行下一步。
- 步骤9** 软件配置寄存器TM_RAM_INIT_ENABLE为0x1。
- 步骤10** 软件查询寄存器TM_INITIAL_DONE；若查询到寄存器TM_INITIAL_DONE为0x1，则软件可以配置系统控制器中的相关寄存器，进行TM软复位，否则继续**步骤6**。
- 步骤11** 配置TM_LGC_RAM_INIT_ENABLE为0x1，初始化内部逻辑的memory。
- 步骤12** 等待2μs以后，软件查询寄存器TM_LGC_INITIAL_DONE；若查询到寄存器TM_LGC_INITIAL_DONE为0x1，可开始执行软件初始化，否则继续**PF初始化的步骤8**，重复查询100次仍未完成则软件上报错误。
- 步骤13** 配置POE_SLV侧到达保序的tm_add_poe_sn号为初始值0x0，保证TM从0开始发送sn号，POE可以正常接收。
- 步骤14** 配置POE/PKIO/EMIO的MST侧的TM对应的sn_cnt和credit，保证各MST的sn号从0开始发送。
- 步骤15** TM解软复位后，软件重新进行PF初始化（跳过**步骤3~步骤8**的初始化memory步骤），VF初始化，初始化完成后，TM可以继续正常处理包任务。

----结束

2.6.9 TM 安全复位释放 Buffer 步骤

- 步骤1** 将POE 推送TM的地址动态改配为一个加包的reserve的地址，此时TM不会返回credit，各MST不会持续泄露Buffer，查询到各MST的credit耗尽时MST即停流。
- 步骤2** SPPE停止入口流量，并保证所有添加到TM的任务都已成功（通路处于空闲状态）。
- 步骤3** 软件配置TM所有物理物理端口不使能TM_PORT_EN。
- 步骤4** 查询TM内部所有队列缓存报文个数TM_FQ_ST_NUM统计，若全为0，则继续**步骤4**，否则重复**步骤3**，如果查询时间超过20ms仍然不为0，则进行下一步。
- 步骤5** 软件配置寄存器TM_MODULE_RST_REQ为0x1。
- 步骤6** 软件查询寄存器TM_MODULE_RST_DONE。



- 步骤7** 软件查询寄存器TM_MODULE_RST_DONE；若查询到寄存器TM_MODULE_RST_DONE为0x1，则继续**步骤7**，否则重复**步骤6**。
- 步骤8** 配置TM所属SUBSYS系统控制器的PPE软复位有效，若TM有部分队列分配给ROCEE（通过TM_ROCE_QUEUE_NUM_CFG配置），则需要软件配置TM所属SUBSYS系统控制器的ROCEE软复位有效。
- 步骤9** 配置寄存器TM_SLV_ACC_WR_PTR_CLR，清除TM_SLV到达保序对应的bitmap和写指针。
- 步骤10** 配置寄存器TM_TK_BIND值为全0，清除表项配置。
- 步骤11** 按照系统复位流程，配置TM所属SUBSYS系统控制器的TM软复位有效，完成复位后进行下一步。
- 步骤12** 软件配置寄存器TM_RAM_INIT_ENABLE为0x1。
- 步骤13** 软件查询寄存器TM_INITIAL_DONE；若查询到寄存器TM_INITIAL_DONE为0x1，则软件可以配置系统控制器中的相关寄存器，进行TM软复位，否则继续**步骤11**。
- 步骤14** 配置TM_LGC_RAM_INIT_ENABLE为0x1，初始化内部逻辑的memory。
- 步骤15** 等待2μs以后，软件查询寄存器TM_LGC_INITIAL_DONE；若查询到寄存器TM_LGC_INITIAL_DONE为0x1，可开始执行软件初始化，否则继续**步骤14**，重复查询100次仍未完成则软件上报错误。
- 步骤16** 配置POE_SLV侧到达保序的tm_add_poe_sn号为初始值0x0，保证TM从0开始发送sn号，POE可以正常接收。
- 步骤17** 复位POE/PKIO/EMIO的MST侧的TM对应的sn_cnt和credit，保证各MST的sn号从0开始发送。
- 步骤18** TM解软复位后，软件重新进行PF初始化（跳过**步骤3~步骤8**的初始化memory步骤），VF初始化，初始化完成后，TM可以继续正常处理包任务。
- 结束

2.7 性能

2.7.1 应用场景

2.7.2 实际测试结果

根据原型测试结果，TM单队列入口性能最小为15Mpps，多队列入口性能最小为20Mpps，大于TM的出口性能（单端口15Mpps/多端口20Mpps）。

2.7.3 吞吐率（Throughput）

需求单端口包率15Mpps，现在按照15Mpps计算其性能，调度一个包需要（1000/15）= 66ns，TM时钟频率600MHz，调度一个包需要的时钟周期数小于：600/15 = 40cycle。

多端口包率20Mpps，现在按照20Mpps计算其性能，调度一个包需要（1000/20）= 50ns，TM时钟频率600MHz，调度一个包需要的时钟周期数小于：600/20 = 30cycle。



将队列调度和队列管理分流水完成，保证调度和队列管理都在40cycle/30cycle完成；

2.7.4 延迟 (Latency)

- 一个报文描述符从添加入TM到被调度出去的延时取决于6个方面：
 - 调度算法本身时延，即SP/DRR调度本身时延。
 - 调度权重的配置值。
 - 调度器层级和调度器的实现。
 - 包描述符的队列管理和存储方式。
 - 总线时延。
 - 发送端口拥塞反压调度。
- TM: 600MHz
 - 包描述符的队列管理和存储采用片内存储方式，时延可忽略。
 - 低优先级队列的延迟和用户的具体shaper、weight配置相关，延迟不可计算。
 - 高优先级队列的延迟主要来源于队列管理，调度和总线延迟。不考虑溢出回读，一个描述符在TM内部延迟大约为300cycle。



3 POE

3.1 概述

POE (Packet Order Enforcer) 模块是Hi309a多核处理器芯片中的系统级消息交互管理模块。支持多个CPU Core之间的消息通信、包保序调度、并发任务调度，并提供硬件IP/HAC的统一消息任务接口，并且支持跨芯片的消息交互和跨DIE转发消息和包任务。

POE中的任务包含两种：

- 包任务，以报文描述符的形式通过POE通讯；
- 核间通讯消息，直接以数据形式通过POE通讯。包任务可在CPU Core间、CPU Core和加速器间、加速器间进行通讯使用，包任务可选择保序，POE只为CPU Core处理的包任务进行保序，不为SEC等加速器处理的包任务进行保序（这些加速器自己负责保序）；核间通讯消息可在CPU Core间进行通信使用，核间通讯消息固定不保序。

按照业务需要，属于同一条流的包需要按照先进先出的顺序进行处理及发送，保证开始处理前的包顺序与处理完被发送的顺序一致。

当多个处理单元并行进行任务处理时，由于每个被处理的包的大小长度不同，每个处理单元处理包的延迟也不一致，先开始处理的包不能够保证最先完成，导致后续处理单元处理完包任务后不能直接发送，要排队等待先开始处理的包任务完成并被发送的情况，造成处理能力的无谓损失，所以多核并行处理包任务时，需要POE这样的硬件保序模块来代替CPU进行排队，降低软件复杂度，释放处理能力，同时保证同流包的顺序。

POE还可以均衡各个处理单元的任务处理负载，按应用需要由软件通过CPU动态配置POE，使得任务可以被指定的一个或多个处理单元处理。

3.2 关键术语

Timer相关的关键术语表如[表3-1](#)所示。



表 3-1 POE 术语表

术语	描述
POE	包保序（ Packet Order Enforcer ）。
QoS	POE调度描述中的QoS指的是优先级，每个队列的QoS优先级都是寄存器可配置，POE按照QoS的值由高到低进行PQ调度。
Group	队列组，4或8个队列为一个Group，只作为资源分配单位，每个处理单元可以处理特定的Group包，一个LCPU可以绑定多个Group，通过Group将包分配到不同处理单元进行处理，达到各处理单元负载均衡的效果。
Qid	在逻辑CPU或加速器添加或获取任务的Queue ID，Qid除以4/8的商即为对应的Group。
缓存队列	POE内部有小缓存来存储处理单元添加到POE的任务包。这个小缓存分成若干个缓存队列，每个队列都是一个先入先出的FIFO。
TT	Tag Type，分为NULL、ORDER、ATOMIC三种类型。
NULL	Tag Type的一种，NULL类型的流没有保序的需求。
ORDER	Tag Type的一种，ORDER类型的流需要按Tag、OG、TT严格保序，可以有多个包任务被多个处理单元并行处理。本芯片中只有CPU core处理报文时需要保序，所以这里的“处理单元”只有CPU core。
ATOMIC	Tag Type的一种，ATOMIC类型的流需要按Tag、OG、TT号严格保序，每条ATOMIC流同时只能有一个包在被处理单元处理。本芯片中只有CPU core处理报文时需要保序，所以这里的“处理单元”只有CPU core。
PPE	Packet Parse Engine（包解析引擎），系统中的硬件包解析引擎，集成在网络接口中，逻辑线速完成包的基本解析、识别工作。
AXI	AMBA的总线协议中的一种，在本文中提到的AXI对应AXI4协议。
处理单元	在本节中，所有能主动对POE进行取包和添加包的模块，都叫处理单元。处理单元可以是实际的物理模块，如物理CPU Core，SEC，SPPE；也可以是逻辑CPU（物理CPU Core虚拟出的逻辑CPU）。
LCPU	物理CPU Core虚拟出的逻辑CPU，最多支持64个逻辑CPU。一个物理CPU Core对应1个或多个逻辑CPU。
真满	POE内部有小缓存来存储添加到POE的任务，如果小缓存满，POE需要把没有进入POE内部小缓存的任务通过总线缓存到DDR中，DDR中分配给POE的缓存空间是软件配置的。如果DDR中分配给POE的缓存也被存满，这时POE就没有空间存储任务了，这时就达到了真满状态。
假满	POE内部有小缓存来存储添加到POE的任务，如果小缓存满，POE需要把没有进入POE内部小缓存的任务通过总线缓存到DDR中，DDR中分配给POE的缓存空间是软件配置的。如果DDR中分配给POE的缓存有空闲空间，但此时POE给DDR传送任务时由于AXI总线的拥塞而写不到DDR，这时就是POE的假满状态。



术语	描述
LPI	Locality-specific Peripheral Interrupt，特指外设中断。
LCPU私有队列	每个LCPU拥有两个私有的任务缓存队列，分别用于PUSH_LCPU和PULL_LCPU，该缓存队列可能在私有SRAM中或者Cache中。
PUSH LCPU	POE从内部队列获取任务后向LCPU私有队列推送任务。
PULL LCPU	POE从LCPU私有Cacheable队列获取任务添加到内部队列。
Cache Stash	POE在LCPU主动获取任务后或者推送任务到LCPU私有队列后，会主动发起L2 Cache的preload操作，预取指定1~17个Cacheline数据，提升业务处理性能。
半静态配置	只允许在模式初始化时修改配置。
静态配置	只允许在设备初始化时刻修改配置。
动态配置	允许任意时刻修改配置。
RR调度	Round Robin，公平轮询各个队列，最简单的调度算法之一。
PQ调度	Priority Queue，基于优先队列的一种调度算法。该算法按照队列优先级的高低进行调度，高优先级先调度，低优先级后调度。高优先级队列非空，就先调度优先级高的队列出队，只有在优先级高的队列调度空后，才调度低优先级队列。
WRR调度	Weighted Round Robin，加权轮询调度。 WRR调度主要解决RR不能设置权重的不足。在轮询的时候，WRR每个队列享受的调度机会和该队列的权重成比例。当所有队列权重为1时相当于RR调度机制（假如权重计数为0停止调度）。
MessageQueue	泰山核固有的消息队列，用于CPU与POE之间任务交互，减少IO时延。

3.3 特性

POE模块具有以下特点：

- 逻辑CPU以128bit/256bit的任务向POE进行添加和获取，加速器以256bit的任务向POE进行添加和获取，支持CPU以128bit的方式添加保序命令。
 - 支持两种混合长度的任务，每个队列只支持一种长度的任务，支持的任务长度软件可静态配置，且任务数据结构包含任务长度的字段。
- 接口特性。
 - 符合AMBA4.0协议。
 - 提供一个Master接口，数据位宽256bit，地址位宽48bit，其中只有向逻辑CPU推送任务和获取任务时32bit/64bit大小端模式可静态配置，其中推送Cacheable队列/从Cacheable队列获取任务的32bit/64bit大小端按模式独立配置，其他固定小端模式。
 - 支持POE向POE添加任务（SEND_POE），128/256bit任务，Size 256bit，Burst1，Outstanding 1~8可静态配置。



- 支持向加速器TM添加任务 (SEND_TM) , 256bit任务, Size 256bit, Burst1, Outstanding 1~8可静态配置。
- 支持向加速器 BMU释放Buffer (SEND_BMU) , 64bit, Burst1, Outstanding 1~8静态可配置。
- 支持转发CPU给TM的任务 (FORWARD_TM) , 256bit任务, Size 256bit, Burst1, Outstanding 1~12可静态配置。
- 支持推送任务到逻辑CPU的Cacheable私有队列, 128/256bit任务, Size 256bit, Burst1, Outstanding 1~64可静态配置。
- 支持从逻辑CPU的Cacheable私有队列获取任务, 128/256bit任务, Size 256bit, Burst1, Outstanding 1~8可静态配置。
- 支持推送给各个加速器任务处理通知消息, 32bit, Burst1, Outstanding 1~16可静态配置。
- 支持溢出DDR 操作256bit, Burst2, Outstanding1~24可静态配置。
- 支持回读DDR 操作256bit, Burst2, Outstanding1~24可静态配置。
- 提供一个Slave接口, 数据位宽256bit, 地址位宽48bit。
- CPU向POE配置寄存器时, 仅支持小端模式; 支持single 32bit操作, Outstanding1; 支持多个CPU并发、无锁访问。
- 支持CPU添加任务时, 若为256bit任务, 支持32bit Burst1~ Burst8, 64bits Burst1~ Burst4, 128bit的Burst1~Burst2操作; 若为128bit任务, 支持32bit Burst1~ Burst4, 64bit Burst1~ Burst2, 128bit的Burst1操作。
- 支持CPU添加/获取任务, 大小端模式寄存器静态配置, 默认为32bit大端, 支持按制式单独配置大小端模式。
- 支持加速器添加任务, 仅支持小端模式; 添加任务的256bit, Burst1操作, 支持对加速器添加任务到达保序。
- 支持加速器获取任务, 仅支持小端模式; 获取任务的256bit, Burst1操作。
- 支持统一的I/O、HAC加速器访问接口。
 - POE有与所有加速器一致的中断地址和中断编号。
 - 支持软件查询包含硬件规格信息的寄存器。
 - 支持软件可读写设备号PF_ID, 功能号VF_ID。
 - 通道接口访问位宽超过CPU Single操作位宽时, 接口支持保证错误不能扩散, 防止错误传播。
 - 采用统一的虚拟化 PF/VF接口。
 - 统计类寄存器支持读清方式, 如有个别统计寄存器应用者不希望读清, 在功能描述中枚举。
- 支持虚拟化、多模应用。
 - 支持基于1个PF+5个VF的6个模式的寄存器访问, 其中各VF的寄存器以1M/64KB地址空间为最小粒度分开, 其中CPU的AXI Slave寄存器空间以1MByte为粒度, 加速器的AXI Slave寄存器空间以64kByte为粒度。
 - 支持按照POE Slave接口接收到的地址进行VF索引, 自动判断出该访问所属VF。
 - 支持基于LCPU或模式的中断上报机制, 能够区分LCPU的就按照LCPU上报中断, 否则按照PF/VF上报中断。



- 支持资源虚拟化，即VF业务的LCPU、Group编号可以（但不限制）是一个VF内部从零开始的紧密排列的编号。
- 支持POE Master口发送时，携带STREAMID以支持SMMU进行VA到PA的转换，每个PF/VF的STREAMID由软件独立静态配置；POE推送加速器操作默认不使用STREAMID。
- 支持各VF模式下的业务进行单独复位。
- POE不支持VM虚拟机机制。
- 支持200个任务队列，支持POE内部任务缓存，当内部缓存满时，溢出到外部DDR，提供Group溢出使能配置，默认禁止溢出。
 - 支持对添加的任务按照队列进行缓存FIFO队列管理，POE入口支持基于任务中的“VQI”解析目的QueueID。
 - 提供队列入队/出队的使能配置，默认使能入队/出队。入队的使能开关按照队列为粒度配置，出队的使能开关按照Group为粒度配置。
 - 提供2560个片内缓存任务缓存空间（256bit），支持所有队列共享片内缓存空间，提供VF最大空间限制、单队列最大空间限制和最小空间保证。
单队列最大空间限制提供16个配置模板，按队列配置，软件可通过配置索引为每个队列选择独立的模板；单队列最小空间保证全局配置（推荐4/8/16/32共4个模板），软件可通过配置索引为所有队列选择一个指定模板。
 - 支持按制式独立分配DDR空间，最多支持5个制式同时做业务，每种制式DDR空间的大小和起始地址可独立寄存器静态配置，每种制式最大配置空间是32MByte。
 - 任务溢出到DDR空间时，所属制式的所有队列共享该制式的DDR空间。
 - 支持溢出的总线Mem_attr和回读的总线Mem_attr可通过寄存器静态配置。
 - 支持溢出的总线QoS和回读的总线QoS可通过寄存器静态配置。
- 支持入口反压：
 - POE为每个Group提供一组反压加速器任务添加的水线配置寄存器POE_GRP_BP_TH，当Group中的任务个数大于等于水线值时，此Group不再接收接口加速器的任务添加，丢弃加速器添加的任务，POE负责释放Buffer；大于等于Group水线时，只反压加速器的任务添加，不影响其他处理单元的任务添加。

须知

由于内部处理时延，当Group内的任务个数达到Group反压水线后，可能存在反压丢包会存在1~4个任务的误差。

- POE缓存队列空间（内部缓存空间和DDR空间）满时，上报真满中断，任务丢弃，统计丢弃次数；若是加速器添加，POE通过总线B通道固定返回OK响应，POE负责释放Buffer；若是CPU添加，POE通过总线B通道固定返回OK响应。
- POE出现假满状态时，反压加速器的添加操作，POE通过总线B通道固定返回OK响应，POE负责主动重传添加任务，若重传超过最大重传次数则丢弃并释放Buffer，并上报中断，统计丢弃计数。
- POE任务队列使能关闭时，反压加速器的添加操作，POE通过总线B通道固定返回OK响应，任务丢弃，上报中断，统计丢弃计数。
- POE出现假满状态时，反压CPU的添加操作，POE通过总线B通道返回OK，POE负责主动重传添加任务，若重传超过最大重传次数则丢弃并释放Buffer，并上报中断，统计丢弃计数。



- POE任务队列使能关闭时，反压CPU的添加操作，POE通过总线B通道返回OK，任务丢弃，上报中断，统计丢弃计数。
- 支持POE对TM的反压，通过硬连线反压TM，最多支持6个Group。
- 支持反压状态可查询，支持反压次数（8bit饱和计数、读清）统计可查询。
- 支持出口反压：
 - 支持添加任务到POE/TM若发生真满时，由推送的目的加速器进行释放Buffer处理。
 - 支持添加任务到POE/TM若发生假满，由推送的目的加速器重试添加操作，重试次数到达配置次数时，目的加速器上报中断，任务丢弃，释放Buffer。
 - 支持释放Buffer到BMU时总线反馈假满，POE重试添加操作，重试次数到达配置次数时，POE上报中断，Buffer丢弃。
 - 支持反压状态可查询，支持反压次数统计（8bit饱和计数、读清）可查询。
- 支持QoS调度管理：
 - QoS支持8个优先级，每个队列都有寄存器可单独配置其优先级，QoS7的优先级最高，QoS0的优先级最低，且队列QoS动态可配置。
 - 提供PQ+RR+WRR的三级任务调度方式，逻辑CPU或加速器发起GET_POE命令获取任务时，POE首先按照QoS值对绑定的所有队列做PQ调度；然后对最高QoS的队列所属Group做RR调度挑选出一个Group；最后从一个Group中以WRR方式把该QoS的队列调度出POE。
 - 提供RR+PQ+WRR的三级任务调度方式，逻辑CPU（仅限于CPU）发起GET_POE命令获取任务时，POE首先对绑定的所有Group做RR调度挑选出一个Group；然后对该Group内最高QoS的所有队列做WRR调度挑选出一个队列，PQ+RR+WRR/RR+PQ+WRR调度算法选择按VF独立配置。
 - POE中每个队列的WRR权重值都有寄存器单独可配置，配置范围0~255（配置为0时该队列将不会有包被调度出来），且队列WRR权重值动态可配置。
 - 支持“流分类”，尽可能的降低不同流间队头阻塞的影响。
 - 针对ATOMIC任务，支持基于集合MASK的互斥调度，支持基于MASK进行两个TAG是否互斥的比对；支持对需要互斥的TAG进行互斥调度（并行执行和串行执行），基于集合MASK的互斥调度只针对ATOMIC任务。
- 支持Group管理和负载均衡。
 - 支持提供16个Group（物理Group编号0~15，每个Group有8个队列）给CPU使用，提供18个Group（物理Group编号16~33，每个Group有4个队列）给加速器使用，加速器和CPU不能使用同一个Group（若要使用同一个GROUP，仅限CPU GET_GROUP获取加速器的GROUP）。
 - 每个处理单元可以处理一个或者多个Group的任务。
 - 逻辑CPU处理的Group可以分配给一个或者多个逻辑CPU处理，但加速器处理的Group不能同时分配给多个加速器处理。
 - 在处理单元处理包的过程中，可以动态的添加/移除处理单元和Group的绑定关系（切换时必须保证CPU没有和包任务绑定）。
- 支持Tag管理和保序。
 - 支持三种Tag类型，分别为NULL、ORDER、ATOMIC。
 - NULL的包不需要保序，逻辑CPU获取任务后，POE不保留此包的信息。逻辑CPU处理完后直接分发到下一流程。
 - ORDER的包需要按流保序，同一时刻，多个逻辑CPU可以并行处理多个属于同一条流的包，先进入芯片的包先分发到下一流程，即使后进入芯



片的包先处理完也要等待，直到前面的包处理完分发出，后面的包才能分发。

- ATOMIC的包需要按流保序， ATOMIC流支持两种任务类型， ATOMIC Serial和ATOMIC Parallel， 且具有以下特性：
 - ATOMIC Serial: ATOMIC流中的串行的原子性任务， 与其他 ATOMIC Serial 和ATOMIC Parallel任务互斥， 同一时刻只能执行一个ATOMIC Serial任务。 ATOMIC Serial只能原子执行， 必须等待前面的ATOMIC Parallel或ATOMIC Serial任务都执行完成后， 串行执行； ATOMIC Serial任务阻塞后面ATOMIC Parallel和ATOMIC Serial任务执行。
 - ATOMIC Parallel: ATOMIC流中的并行的原子性任务， 与其他 ATOMIC Serial任务互斥， 但可与其他ATOMIC Parallel任务并行处理， 同一时刻在无ATOMIC Serial任务阻塞时， 可以并行处理多个 ATOMIC Parallel任务。
- Tag号位宽为24 bits。
- 在包处理过程中， 严格按照VFID、 TT、 OG和Tag号来保序， 其中OG按Group可静态配置。
- POE内部支持最多128个保序流， 每个保序流可以动态的产生和消亡， 可重复利用， 这128条流共享128个包任务保序存储单元（ 位宽256bits ）， 保序资源用完时， CPU不能再获取到保序的Group中的任务。
- 提供按VF的保序满水线， 保序任务个数到达VF保序水线时， CPU不能再获取该VF的保序任务。
- 支持128/256bits包任务保序， 核间消息任务固定不保序。
- 支持保序队列队头超时， 超时上报中断但不丢弃队头， 超时时间软件可静态配置。
- 支持ATOMIC Parallel和ATOMIC Serial混合流， 支持一个任务中的若干子任务的多核并发， 并发阶段的子任务是ATOMIC Parallel， 串行的子任务 ATOMIC Serial。
- 支持跨Group和队列的ATOMIC流保序（ 相同Tag和TT的包会入不同的队列， 但Group的配置OG相同 ）， 不同队列之间不保证调度ORDER类型任务的执行顺序， 只保证ATOMIC任务的原子性执行顺序。
- CPU的任务Group支持保序可配置。
 - 提供寄存器对CPU的任务Group进行保序/不保序静态配置。
 - 加速器的任务Group固定不保序。
- 支持消息通讯。
 - 支持核间消息通讯。
 - 支持逻辑CPU将消息发送到指定队列入队。
 - 支持8个QoS优先级的消息通讯。
 - 支持Group中4/8个队列的QoS优先级任意配置。
 - 支持逻辑CPU指定队列入队， 指定Group出队。
 - 支持128/256bits的核间消息通讯。
 - 支持消息任务队列和包任务队列混合配置到同一个Group内进行调度处理。
- 提供Event脉冲信号给各个物理CPU， 以支持物理CPU可以通过WFE方式被唤醒。
 - POE Group绑定多核场景下， 支持WFE唤醒物理CPU的策略可配置。



- 提供中断给各个逻辑CPU，以支持物理CPU可以通过WFI方式被唤醒。
 - POE Group绑定多核场景下，支持WFI唤醒CPU的策略可配置。
- 支持转发CPU给TM的任务（FORWARD_TM）。
 - 转发的任务根据VF分别缓存在不同的队列中。
 - 每个VF支持1个转发任务队列，每个队列仅支持256bits任务的添加和转发。
 - 转发任务队列按VF共享片内缓存空间和片外DDR溢出空间，支持溢出回读，支持到达保序。
- 支持CPU和加速器添加/获取任务。
 - 支持最多16个逻辑CPU通过访问POE内部寄存器的方式向POE添加任务、读取任务、或指示POE发送包任务；支持加速器通过访问POE内部寄存器的方式向POE添加任务、读取任务，或指示POE发送包任务。
 - 为每个逻辑CPU提供一个独立的添加任务的（ADD_PACKET命令）寄存器接口。
 - 为每个逻辑CPU提供一个独立的依照调度算法读取任务的（GET_POE命令）寄存器接口。
 - 为每个逻辑CPU提供一组独立的指定队列读取任务的（GET_QUEUE命令）寄存器接口。
 - 为每个逻辑CPU提供一组独立的指定Group读取任务的（GET_GROUP命令）寄存器接口。
 - 为每个向POE添加包任务的加速器，每个VF提供一个独立的添加包任务的（ADD_PACKET命令）寄存器接口。
 - 为每个从POE获取包任务的加速器，每个VF提供一个独立的依照调度算法读取任务的（GET_POE）寄存器接口。
 - 5个VF分别有一套上述的寄存器接口。
- 支持向逻辑CPU推送（PUSH_LCPU）任务和向逻辑CPU获取（PULL_LCPU）任务。
 - 支持推送核间消息任务或包任务到Cacheable的存储空间，支持针对Cacheable的存储空间的队列的轮询和反压机制（仅限GET_POE操作的任务调度方式）。
 - 支持POE轮询Cacheable的主存空间，任务有效时，POE从Cache获取CPU要添加的任务，轮询间隔软件可静态配置。
- 支持正常业务中断。
 - 支持收包中断，队列中任务超过配置门限，或队列超时无取包行为时产生该脉冲，按照Group汇聚并上报脉冲中断；是否有包状态和是否超时的状态可查询。
 - 支持每个队列中任务超过高水线中断，高于高水线之后又下降到低水线之下的中断，这两种中断按LCPU为粒度上报，提供队列水线状态软件可查询。
 - 支持每个Group中任务超时未处理上报中断，按LCPU为粒度上报。
 - 支持每个Group中任务超过Group水线上报中断，按LCPU为粒度上报。

以下是业务DFX特性：

- 支持逻辑CPU查询任务与该逻辑CPU绑定关系。
- 支持异常中断。
 - 针对逻辑CPU独立上报中断；无法归类到LCPU的中断，分VF或PF上报中断。



- 物理队列粒度的满中断，CPU所属队列按LCPU为粒度上报，加速器所属队列按VF为粒度上报。
- 支持每个制式中任务超过制式水线上报中断。
- 支持包保序队列队头超时未出队上报中断，支持超时门限寄存器静态配置。
- 访问DDR时总线返回非OK的中断上报。
- 内部所有非寄存器搭建的memory支持 ECC 1bit、ECC 2bit错误检测和中断上报，并进行出错饱和统计，所有ECC 1bit错误使用一个统计计数器一个，所有ECC 2bit错误使用一个统计计数器。
- 内部ECC 1bit通过SPI中断发送给GIC；内部ECC 2bit通过SPI中断以致命中断发送给GIC，其他中断也通过SPI中断发送给GIC。
- 支持逻辑CPU添加任务到POE出现真满时，丢弃任务，总线反馈OK给逻辑CPU，上报中断并且记录中断信息（源逻辑CPU），并支持反压历史状态查询，反压次数查询。
- 支持加速器添加任务到POE出现真满时，丢弃任务，总线反馈OK给加速器，并上报中断并且记录中断信息（源加速器），并支持反压历史状态查询，反压次数查询。
- 支持加速器添加任务到POE 时出现SN错误，总线反馈OK给加速器，并上报中断并且记录中断信息（源加速器）。
- 支持Debug Bus接口，可访问POE中所有寄存器。
- 支持软件只读访问POE内部部分RAM空间。
- 支持同时监控所有Group的add和get次数，其中get不需统计无效包。
- 支持CPU对POE的内部队列和其他FIFO的空满状态查询。
- 支持ECC故障注入和中断注入。
- POE增加时戳，能够统计报文/消息在POE内的latency，时戳精度为可配置，默认为0.1μs，宽度为24bit。

3.4 遵循协议

POE模块接口遵循AMBA4.0协议。

3.5 功能描述

3.5.1 POE 应用

POE在整个Hi309a芯片中的应用如图3-1所示。

POE可以作为整个系统的任务交互中心，所有处理单元都可以通过POE获取任务。应用图中通路介绍如下：

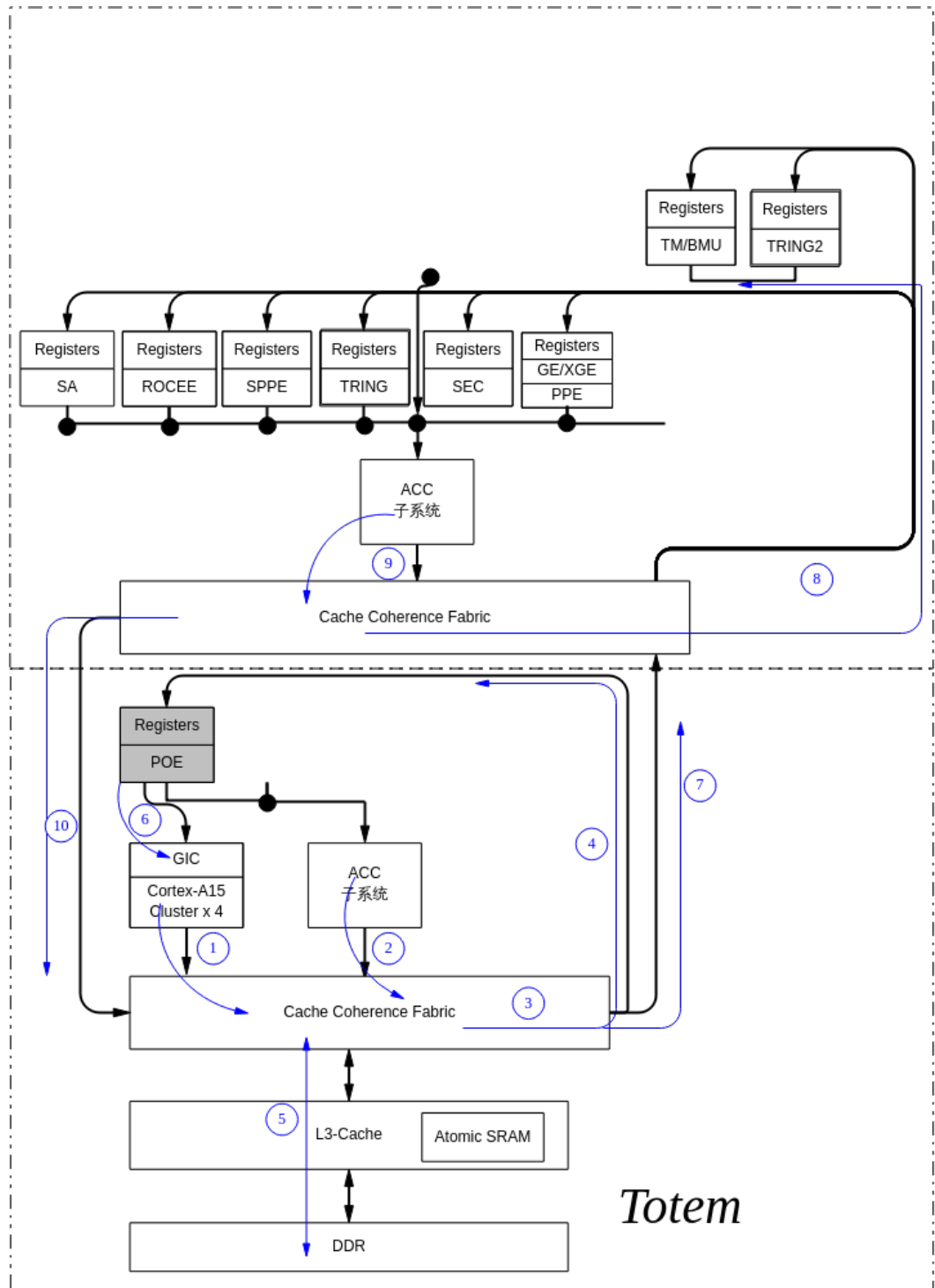
- CPU Core通过①、③、④线路配置POE、从POE获取任务、添加任务到POE中、或对POE中包进行发送保序命令操作。
- ROCEE、SPPE、SEC、TRING、SA通过⑨、⑩、③、④线路从POE获取任务、或添加任务到POE中。
- TM、通过⑨、⑩、③、④线路添加任务到POE中。
- PPE通过⑨、⑩、③、④线路添加任务到POE中。



- POE通过②、③、⑦、⑧线路将任务推送到TM。
- POE通过②、③、④线路将任务推送到同DIE的POE。
- POE通过②、③、⑦、⑧线路通知、SPPE、SEC、TRING、SA，指示HAC VF绑定POE Group有包到达。
- POE通过②和⑤线路对DDR进行读写。
- POE通过⑥线路上报中断。

POE通过②、③、⑦、⑧线路到BMU释放Buffer。

图 3-1 POE 应用示意图



3.5.1.1 相关模块

在系统中，POE模块会和下面多个模块进行交互，分别描述如下表3-2所示。



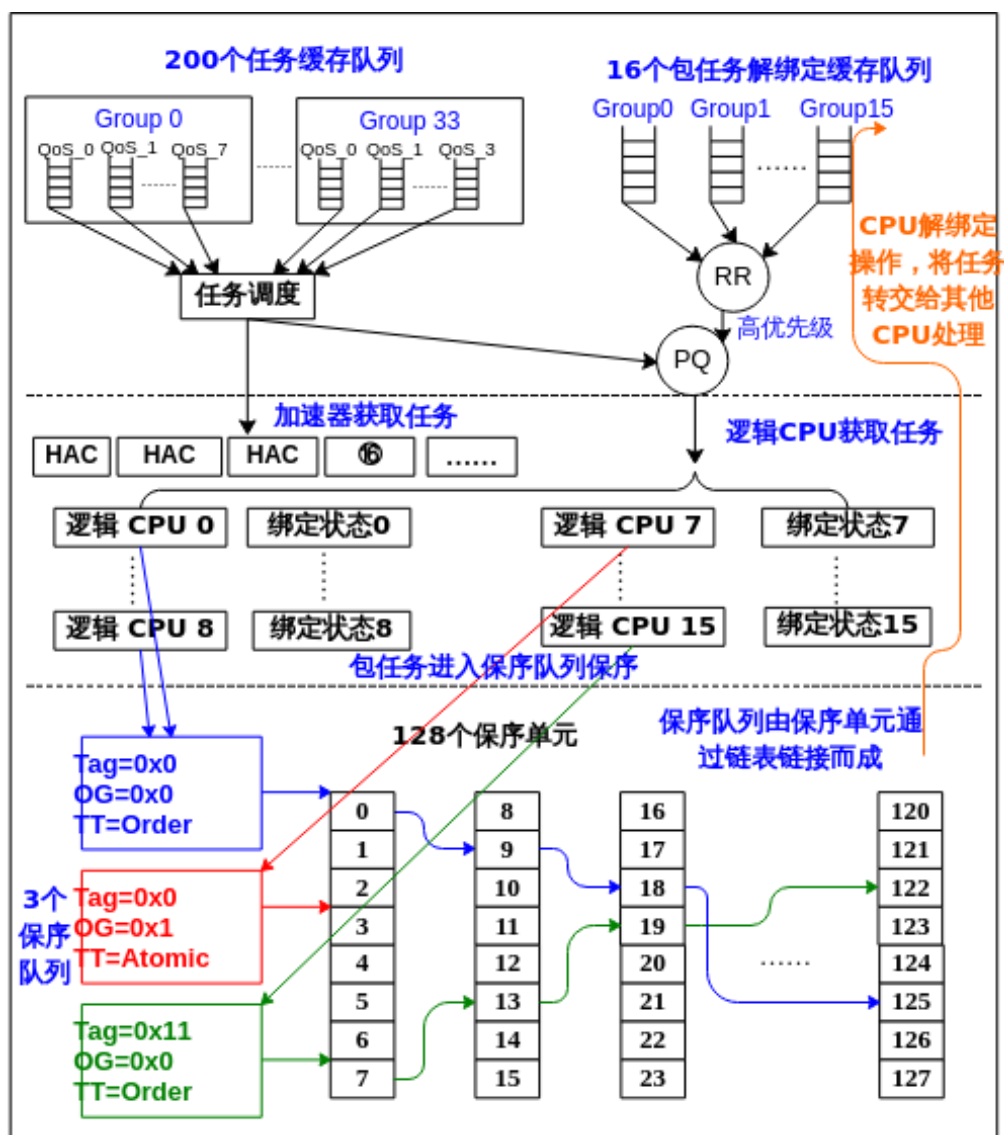
表 3-2 系统中各个模块操作 POE 列表

Hi309a加速器	ADD_POE (Outstanding深度)	GET_POE (Outstanding深度)
PPE	是 (8)	否
PPE_1588	是 (1)	否
SEC	是 (16)	是 (16)
TM	是 (8)	否
SA	是 (2)	是 (2)
ROCEE	是 (24)	否
TRING	是 (1)	是 (1)
SPPE_PKIO	是 (8)	是 (8)
SPPE_EMIO	是 (8)	是 (8)
POE (SEND_POE, ACC_ID=17)	是 (8)	否

- PPE: PPE通过系统总线将包任务添加到POE中, POE接收PPE解析后的包任务。
- CPU Core: CPU Core通过系统总线从POE中获取任务, 或将处理完的任务添加回POE。在处理过程中, CPU Core可以通过POE特有保序命令与POE进行交互, POE完成多CPU Core之间包任务的保序; POE通过总线推送任务到LCPU私有队列和从LCPU私有队列获取任务。
- SPPE: SPPE通过系统总线从POE中获取包任务, 或将处理完的包任务添加回POE。
- SEC: SEC通过系统总线从POE中获取包任务, 或将处理完的包任务添加回POE。
- TRING: TRING通过系统总线将包任务添加到POE中, 或通过系统总线从POE中获取包任务。
- TM: POE通过系统总线将完成的保序包任务推送给TM, TM再将包任务通过网口发出去; TM也会通过系统总线将特定包任务添加到POE。
- DDR: POE内部缓存满时, 溢出任务到DDR中; 当POE内部缓存一个cacheline的空间时, 将任务从DDR中读回来。
- POE: POE依据保序命令将保序包任务通过系统总线重新添加回POE缓存队列中。
- BMU: 加速器/CPU添加POE时, 若发生真满或者假满重传超时, 则向BMU释放Buffer。

3.5.1.2 模块框图

图 3-2 POE 内部工作示意图



- 逻辑CPU向配置的 Group队列进行任务获取，包任务可选择配置为保序，在包任务处理结束后可通过解绑定命令，将包任务放入“解绑定缓存队列”中，转交给其他逻辑CPU处理。逻辑CPU在向POE获取任务时，“解绑定缓存队列”的任务比“任务缓存队列”的任务有高优先级被逻辑CPU获取处理。
- 加速器向“任务缓存队列”获取任务，任务不在POE中保序。加速器处理结束后可添加回POE不同的“任务缓存队列”中，让下个处理单元来获取此任务进行下阶段的处理。
- 各个处理单元之间对包进行Pipeline（任务的分步骤流水处理）处理时，POE可以起到中转路由的功能。一个处理单元将包处理完后，可以重新将包添加到POE不同的“任务缓存队列”中，让下个处理单元来获取此任务进行下阶段的处理。



3.5.2 CPU 访问寄存器

访问POE的寄存器时，仅支持小端模式；支持single 32bits操作。

3.5.3 CPU 加包 (ADD_PACKET)

逻辑CPU通过ADD_PACKET命令添加包任务到POE中。POE通过总线B通道返回OK，由上层CHI总线通过CompData返回OK或真满状态给CPU。

- VF制式添加包任务

POE为VF0~VF4制式都提供一组共16个虚拟CPU的添加包任务
VCPU_WORD_WR_CMD寄存器接口。

其中，VF添加包任务到POE时，POE根据任务添加的地址译码到添加操作的VFID和VCPU，根据“[逻辑CPU资源的虚拟化](#)”的映射关系得到当前操作的LCPU。

POE基于任务中的“VQI”解析目的QueueID，POE为每个VF单独提供一个深度为1024/宽度为10的VQI路由表，路由表每个表项存储VQI和虚拟Qid的映射关系，支持通过添加操作的VFID和数据结构中的VQI，访问特定VF的VQI路由表的偏移为当前VQI的地址，获取虚拟Qid，并完成虚拟Group到物理Group的转换。软件可通过配置使能POE_VQID_ROUTE_BYPASS_CFG，直接使用包任务数据结构中的虚拟VQI作为虚拟Qid，并完成虚拟Group到物理Group的转换。

POE将包任务添加到对应的缓存队列后，更新对应物理Group的统计计数和对应逻辑CPU的统计计数。

上述对应逻辑CPU和物理Group的计数器都是循环计数，软件读清。各个VF的软件读取对应VF的VCPU_ADD_CNT和GRP_ADD_CNT，获取到对应VCPU和虚拟Group的加包统计值。

另外，各制式加包成功后，各制式对应的CURRENT_PKT_CNT进行加1操作。

须知

CPU添加任务时，若发生真满或添加假满重传超时，任务会写入“任务失败缓存”中。

3.5.4 CPU 添加保序命令

逻辑CPU通过SWITCH_NULL / SWITCH_DESCCHEDULE/ DESCCHEDULE /SEND_TM/ SEND_POE命令添加保序任务到POE中。逻辑CPU添加，POE通过总线反馈OK。

- VF制式添加包任务

POE为VF0~VF4制式都提供一组虚拟CPU的添加保序命令
VCPU_WORD{0~15}_WR_CMD寄存器接口。

其中，VF添加包任务到POE时，POE根据任务添加的地址译码到添加操作的VFID和VCPU，根据“[逻辑CPU资源的虚拟化](#)”的映射关系得到当前操作的LCPU。

须知

CPU添加保序命令时，不会发生假满和真满。



3.5.4.1 SWITCH_NULL

包任务释放命令，逻辑CPU通过此命令通知POE该逻辑CPU绑定的包任务不需要再保序，此包任务在保序队列队头时，POE直接将此包任务信息从保序队列中删除。

须知

Hi309a产品的GET_POE命令之前是否有SWITCH_NULL不做要求。

3.5.4.2 SWITCH_DESCCHEDULE

包任务切换、解绑定命令，逻辑CPU通过此命令改变该逻辑CPU绑定的包任务的域值，并将包任务从当前的保序队列切换到目的保序队列；解绑定操作就是POE将包任务与逻辑CPU解除绑定关系，包任务信息能够被其他逻辑CPU获取进行处理，但包任务信息仍在保序队列中保序。解绑定的包任务只有在SWITCH操作完成后才能被其他逻辑CPU获取，并且解绑定的包任务相较于其他任务有最高优先级被逻辑CPU获取。

须知

- SWITCH_DESCCHEDULE命令设置TT为NULL，等同于SWITCH_NULL命令。
- SWITCH_DESCCHEDULE命令设置TT为ATOMIC，等同于SWITCH_NULL命令。
- SWITCH_DESCCHEDULE不支持切换GROUP/QUEUE/VQI操作，如有应用场景可以通过SEND_POE实现其功能。
- SWITCH_DESCCHEDULE不支持TT从ORDER切ATOMIC操作，如有应用场景可以通过SEND_POE实现其功能。
- SWITCH_DESCCHEDULE不支持TT从ATOMIC切ORDER操作，如有应用场景可以通过SEND_POE实现其功能。

3.5.4.3 DESCCHEDULE

包任务解绑定命令，单独执行解绑定操作，POE将包任务信息与当前逻辑CPU解除绑定关系，包任务信息能够被其他逻辑CPU获取进行处理，但包任务信息仍在保序队列中保序。

3.5.4.4 SEND_TM/SEND_POE

包任务转发命令，逻辑CPU处理完包任务后，并且包任务在POE中保序，逻辑CPU发送此命令给POE，POE收到这条保序命令后，会解除此包任务和该逻辑CPU的绑定关系。POE判断此包是否在保序队列队头，在队头就将包发送给TM或添加回POE；如果包不在保序队列队头，就等包到达队头时执行发送操作。逻辑CPU发送此命令后就可以去获得新包进行处理。

POE的SEND命令包任务转发地址由POE中PF制式的寄存器POE_SEND_*_VF_ADDR_H_CFG、POE_SEND_*_VF_ADDR_L_CFG配置。“*”代表TM、POE名称。

每个加速器有一组VF地址偏移配置寄存器POE_SEND_*_VF_OFFSET（每个VF的偏移值有效位宽为3bit，表示该VF地址对应与VF0地址，偏移的VF空间个数）和加速器使



用VF空间大小配置寄存器POE_SEND_*_STRIDE_SIGN。POE通过将POE_SEND_*_VF_OFFSET寄存器配置的对应制式偏移值，替代POE_SEND_*_VF_ADDR_L_CFG寄存器配置地址中的对应位置，产生各个制式的发送地址。

若POE_SEND_*_STRIDE_SIGN寄存器配置为“1”（即对应加速器的VF空间使用2M空间），将POE_SEND_*_VF_OFFSET寄存器中对应制式的偏移值替换POE_SEND_*_VF_ADDR_L_CFG寄存器中bit[23:21]，若POE_SEND_*_STRIDE_SIGN寄存器配置为“0”（即对应加速器的VF空间使用64K空间），将POE_SEND_*_VF_OFFSET寄存器中对应制式的偏移值替换POE_SEND_*_VF_ADDR_L_CFG寄存器中bit[18:16]。

须知

- SWITCH_NULL、SWITCH_DESCCHEDULE、DESCCHEDULE、SEND_TM、SEND_POE命令是针对在POE保序队列中保序的包任务。这些命令只有逻辑CPU才能使用，其他加速器不会使用。
- 逻辑CPU在成功添加命令SWITCH_NULL、SWITCH_DESCCHEDULE、DESCCHEDULE、SEND_TM、SEND_POE到POE后，可立即再次向POE获取任务处理，不用等待命令完成。
- 没有已绑定保序任务的逻辑CPU，禁止使用上述命令。
- SEND_TM转发功能支持转发1个TM，由数据结构中“Location”域段值确定推送的目的TM。若Location为0，向寄存器POE_SEND_TM_*_ADDR_*_CFG_0对应的TM推送。
- SEND_POE转发功能支持转发1个POE，由数据结构中“Location”域段值确定推送的目的POE。若Location为0，向寄存器POE_SEND_POE_*_ADDR_*_CFG_0对应的POE推送。
- 按照CHI协议，POE_SEND_TM_*_ADDR_*_CFG_*需要配置为64Byte对齐。

3.5.5 CPU 添加转发 TM 任务

FORWARD_TM硬转发命令，该命令用于CPU向TM推包时，将POE当做中转，CPU将包任务推送给POE，POE首先将包任务存储到到专用硬转发队列中，为TM提供每个VF各1个队列，共5个队列，由POE将包直接转给TM。

每个VF的队列与内部缓存队列共享该VF内部缓存空间、片内溢出空间和片外DDR溢出空间，其中内部缓存空间、片内溢出空间提供单队列最大空间限制，单队列最大空间按队列配置，软件只需配置模板索引POE_FWD_Q_MAX_INDEX；FORWARD_TM的队列固定使用选定的最小值模板的2倍值，按队列配置最小空间保留使能POE_FWD_Q_MIN_RSV_EN。

逻辑CPU添加的FORWARD_TM的操作同样是通过ADD_PACKET接口进行任务添加。逻辑CPU添加，POE通过总线反馈OK。

POE的FORWARD_TM命令包任务发送地址由POE中PF制式的寄存器POE_FORWARD_*_VF_ADDR_H_CFG、POE_FORWARD_*_VF_ADDR_L_CFG配置。“*”代表TM名称。每个加速器有一组VF地址偏移配置寄存器POE_FORWARD_*_VF_OFFSET（每个VF的偏移值有效位宽为3bit，表示该VF地址对应与VF0地址，偏移的VF空间个数）和加速器使用VF空间大小配置寄存器POE_FORWARD_*_STRIDE_SIGN。POE通过将POE_FORWARD_*_VF_OFFSET寄存器配置的对应制式偏移值，替代POE_FORWARD_*_VF_ADDR_L_CFG寄存器配置地址中的对应位置，产生各个制式的发送地址。



若POE_FORWARD_*_STRIDE_SIGN寄存器配置为“1”（即对应加速器的VF空间使用2M空间），将POE_FORWARD_*_VF_OFFSET寄存器中对应制式的偏移值替换POE_FORWARD_*_VF_ADDR_L_CFG寄存器中bit[23:21]，若POE_FORWARD_*_STRIDE_SIGN寄存器配置为“0”（即对应加速器的VF空间使用64K空间），将POE_FORWARD_*_VF_OFFSET寄存器中对应制式的偏移值替换POE_FORWARD_*_VF_ADDR_L_CFG寄存器中bit[18:16]。

须知

FORWARD_TM转发功能支持转发1个TM，由数据结构中“Location”域段值确定推送的目的TM，若Location为0，向寄存器POE_FORWARD_TM_*_ADDR_*_CFG_0对应的TM推送。

按照CHI协议，POE_FORWARD_TM_*_ADDR_*_CFG_*需要配置为64Byte对齐。

POE中有对应每个制式的添加FORWARD_TM的计数统计，发送FORWARD_TM任务描述符个数的计数统计。每个制式的寄存器对应为FORWARD_TM_REC_CNT、FORWARD_TM_SEND_CNT，这些计数器寄存器都是循环计数并读清。

3.5.6 加速器加包（ADD_PACKET）

加速器通过ADD_PACKET命令添加包任务到POE中。加速器添加，POE通过总线反馈任务添加状态，反馈状态OK。

- VF制式添加包任务

POE为每个VF制式都提供一组添加包任务接口：ACC_PKT_ADD。POE根据任务添加的地址译码到VF和ACC ID，将任务添加到对应的物理队列中。

POE将包任务添加到对应的缓存队列后，更新对应物理Group的统计计数和对应ACC的统计计数。

按制式统计各加速器成功添加到POE中的任务个数，循环计数，读清；参看寄存器ACC_ADD_CNT等。

另外，各制式加包成功后，各制式对应的CURRENT_PKT_CNT进行加1操作。

- 加速器添加POE失败处理：

加速器添加任务到POE时，可能会添加失败，POE的失败原因和上报方式如下：

- 真满：如果是片内缓存队列和DDR的空间全满，不能添加任务，丢弃任务，上报中断，寄存器POE_HAC_ADD_REAL_FULL_INFO记录加速器添加真满的历史状态，POE_HAC_ADD_REAL_FULL_CNT寄存器记录所有加速器添加真满的次数；
- PPE添加任务的GROUP任务数达到水位：丢弃任务，不上报中断，寄存器HISTORY_BP_PPE_CNT按制式记录PPE历史反压状态；
- 假满：总线拥塞，POE来不及溢出，DDR还有空间，但不能添加任务，POE自动重试添加任务，重试直到超时或者真满丢弃；
- 队列禁止入队：上报队列禁止入队错误中断给对应制式的VF中断，任务丢弃，各制式分别记录加速器添加的历史状态POE_HAC_QIN_ERR_INFO寄存器，记录添加失败次数POE_HAC_ADD_QIN_ERR_CNT；
- VQI路由Queue ID失败，POE不接收该任务的添加，任务丢弃，上报中断给对应制式的VF中断，按制式记录添加失败的加速器历史状态POE_HAC_GQ_ERR_INFO寄存器，记录添加失败的次数POE_HAC_ADD_GQ_ERR_CNT寄存器。



须知

加速器添加POE的outstanding深度若超过了POE最大的预留深度，可能导致不可预知的异常，需要软件在配置加速器添加POE的outstanding深度配置寄存器时保证配置正确。

3.5.7 CPU 获取包

3.5.7.1 命令类型

- GET_POE

处理单元获取任务命令。GET_POE命令首先从“解绑定缓存队列”中获取任务，若其中没有对应任务，则到流分类队列和内部缓存队列去采用QoS调度，获取得到任务。处理单元通过此命令向POE获取包任务/消息，一旦获取任务成功，POE反馈任务给处理单元。如果POE无法提供任务给该处理单元，POE反馈给处理单元的数据“SourcePort[6:0]”域值为“0x7F”（即获取任务失败标识），CPU过段时间再向POE获取任务。如果此任务是包任务且配置为保序，包信息入POE保序队列，并且包任务和CPU建立绑定关系；如果此任务不需要保序，任务信息不入POE保序队列。

- GET_GROUP

CPU指定POE的Group获取包任务/核间消息。GET_GROUP首先从该GROUP的“解绑定缓存队列”中获取，若其中没有对应任务，则到流分类队列和内部缓存队列的对应Group去获取。处理单元通过此命令向POE获取包任务/消息，一旦获取任务成功，POE反馈任务给处理单元。如果POE无法提供任务给该处理单元，POE反馈给处理单元的数据“SourcePort[6:0]”域值为“0x7F”（即获取任务失败标识），CPU过段时间再向POE获取任务。如果此任务是包任务且配置为保序，包信息入POE保序队列，并且包任务和CPU建立绑定关系；如果此任务不需要保序，任务信息不入POE保序队列。

须知

使用GET_GROUP命令，可以获得所有Group的任务，包含加速器的所有Group。

- GET_QUEUE

CPU指定POE队列获取包任务/核间消息。GET_QUEUE命令，到内部缓存的对应Queue去获取。处理单元通过此命令向POE获取包任务/消息，一旦获取任务成功，POE反馈任务给处理单元。如果POE无法提供任务给该处理单元，POE反馈给处理单元的数据“SourcePort[6:0]”域值为“0x7F”（即获取任务失败标识），CPU过段时间再向POE获取任务。



须知

- 注意1: 使用GET_QUEUE命令获取到的包, POE内都不进行保序。无论此包的Group配置为是否保序, 都不起作用。
使用GET_QUEUE命令, 可以获得所有队列的任务, 包含加速器的所有队列。
已绑定保序任务的CPU如果再发送GET_POE、GET_GROUP命令给POE, 反馈获取任务失败。
- 注意2: 当GET_POE/GET_GROUP/GET_QUEUE读取的任务中, 域SourcePort[6:0]的值是7F时, 表示此任务是无效任务。
POE反馈SourcePort[6:0]=7F的场景如下:
 - 逻辑CPU添加7F任务到POE队列中, 被其他处理单元读取到。
 - POE队列中队列空时或没有成功调度出任务时会反馈7F给处理单元。
 - POE从DDR回读任务时, 若响应为error, 则此时读数据也是错误的, POE会将读数据置为7F无效的任务, 并上报总线error中断。
 - 配置POE中Group的出队使能为禁止时, 读取此Group时, POE会反馈7F任务。
 - 当某制式下保序空间使用达到水线, 或者所有的保序空间已经都被使用, CPU获取任务会是7F任务。
 - CPU配置加速器中SourcePort为7F, 导致该加速器添加到POE的任务是7F的任务。
 - Group的虚拟到物理映射映射异常/LCPU绑定异常/LCPU映射异常, 读取此Group时, POE会反馈7F任务。
 - ATOMIC包原子性冲突, POE会反馈7F任务。
 - 队列缓存ECC 2bit错误, POE会反馈7F任务。
 - 上述场景中除了添加到POE的SourcePort为7F的任务, 其他场景POE会将对应无效任务的TLV置为0xFF。

3.5.7.2 命令处理

VF制式获取包任务

- GET_POE
POE为每个VF制式都提供一组虚拟CPU获取任务的GET_POE寄存器接口VCPU_WORD_GET。
POE根据读操作地址译码得到当前GET_POE操作的VFID和VCPU, 由VCPU从对应的制式的配置寄存器GRP_VCPU_MASK0~2得到能获取任务的虚拟Group组, 并通过Group资源的虚拟化的映射关系, 最终得到当前虚拟CPU的GET_POE操作能获取的物理Group组, 通过三级调度, 得到读取的任务反馈给对应的虚拟CPU。
- GET_GROUP
POE为每个VF制式都提供一组虚拟CPU指定虚拟Group获取任务的GET_GROUP寄存器接口VCPU_GRP_WORD_GET。
POE根据读操作地址译码得到当前GET_GROUP操作的VFID和虚拟Group编号, 通过Group资源的虚拟化的映射关系, 得到指定的物理Group编号, 通过三级调度, 得到读取的任务反馈给对应的虚拟CPU。



- GET_QUEUE

POE为每个VF制式提供一组虚拟CPU指定队列获取任务的GET_QUEUE寄存器接口VCPU_Q_WORD_GET。

POE根据读操作地址译码出当前GET_QUEUE操作的VFID和虚拟队列编号，根据虚拟队列与虚拟Group的关系得到虚拟的Group号和Qid，通过Group资源的虚拟化的映射关系，得到指定的物理Group编号，POE根据物理Group编号和Qid，到指定的缓存队列读取任务反馈给对应的虚拟CPU。

3.5.8 加速器获取包

VF制式获取包任务

- POE为每个VF制式都提供一组包含所有向POE获取任务的加速器的GET_POE寄存器接口：ACC_PKT_GET。
- POE根据读操作地址译码得到当前GET_POE操作的VFID和加速器编号，由加速器的对应制式的虚拟Group绑定关系，得到能获取任务的虚拟Group组，并通过Group资源的虚拟化的映射关系，最终得到当前加速器的GET_POE操作能获取的物理Group组，通过三级调度，得到读取的任务反馈给对应的加速器。

须知

SPPE作为一个特殊的加速器，在使用GET_POE命令时，只可能获取物理加速器的包任务，如果SPPE需要在所有缓存队列中获取包任务，可以假冒CPU发送GET_QUEUE命令给POE以获取指定队列的包任务。

3.5.9 入口反压功能

详见《Hi309a 通用MPU芯片 用户指南01-CPU小系统》中的11章节系统反压和流量控制。

3.5.10 出口反压功能

详见《Hi309a 通用MPU芯片 用户指南01-CPU小系统》中的11章节系统反压和流量控制。

3.5.11 统一的加速器访问接口

3.5.11.1 与所有加速器一致的中断地址和中断编号

这个在寄存器中的地址排列体现，即各个加速器按照统一的寄存器排列规则进行寄存器排列。

3.5.11.2 软件可查询硬件规格信息

POE提供硬件规格信息软件可查询，通过寄存器POE_DEVICE_ID查询厂商ID、POE_REVISION_ID查询加速器ID，POE_CACHE_LINE_SIZE查询cache line的大小。详细说明参见《Hi309aV100 POE寄存器描述》。



3.5.11.3 软件可读写设备号 PF_ID，功能号 VF_ID

POE提供设备号PF_ID，功能号VF_ID软件通过寄存器POE_PF_ID和POE_VF_ID可读写访问。

3.5.11.4 采用统一的虚拟化 PF/VF 接口

POE给各VF制式都提供统一的虚拟化接口，各VF制式都有完整的添加/获取任务接口，POE将物理资源通过虚拟化映射，给每个种制式都提供完整的虚拟资源管理接口，每种制式的添加/获取任务的操作虚拟编号（虚拟CPU编号、虚拟Group编号）都可以从0开始。

3.5.12 支持多模及资源的虚拟化

POE支持1个PF和5个VF；PF（Physical Function，物理功能）管理所有VF的公共功能以及各个VF之间的资源分配。VF（Virtual Function，虚拟功能）是能够独立完成业务处理的功能（如与队列相关的配置寄存器，运行过程需要频繁操作的接口寄存器等）。

每个制式都有一套独立的Group资源、CPU资源和寄存器资源，POE中Group资源、CPU资源，通过虚拟化的方式分配给各制式使用。

3.5.12.1 Group 资源的虚拟化

POE为CPU提供了16个的物理Group资源。

软件通过配置寄存器POE_GRP_VP_MAP和POE_GRP_VF_MAP将各制式所使用的V_Group与P_Group进行映射。例如通过配置寄存器POE_GRP_VF_MAP，将P_Group0、P_Group1、P_Group14、P_Group15配置成VF0属性，配置寄存器POE_GRP_VP_MAP，将P_Group0配置给V_Group0、P_Group1配置给V_Group1、P_Group14配置给V_Group2、P_Group15配置给V_Group3；同样的其他P_Group与V_Group的映射关系通过寄存器POE_GRP_VP_MAP和POE_GRP_VF_MAP进行配置。通过配置P_Group与V_Group的映射关系，将物理的Group资源虚拟化到各个制式中。

3.5.12.2 逻辑 CPU 资源的虚拟化

POE支持16个逻辑CPU的访问操作。

软件通过配置寄存器POE_CPU_VL_MAP和POE_CPU_VF_MAP将各制式所使用的V_CPU与LCPU进行映射。例如通过配置寄存器POE_CPU_VF_MAP，将LCPU0、LCPU1、LCPU15配置成VF0属性，通过配置POE_CPU_VL_MAP寄存器将LCPU0配置给V_CPU0、LCPU1配置给V_CPU1、LCPU15配置给V_CPU2，通过寄存器的配置将逻辑CPU的虚拟化属性都进行配置。

3.5.12.3 寄存器资源的虚拟化

POE的寄存器资源可分为模块级公共寄存器空间、PF访问VF0~4相关寄存器空间和VF0~4制式各自业务寄存器空间。

- 模块级公共寄存器：
POE中模块级公共寄存器PF具有读写访问；这些寄存器映射到VF0~4的各自寄存器地址空间，各个VF只有读权限。
- VF制式工作寄存器：



VF制式工作寄存器分为制式粒度寄存器、逻辑CPU粒度寄存器、Queue粒度寄存器和Group粒度寄存器。

- 制式粒度寄存器：每种制式都有一套独立的物理资源，通过地址译码出是那种制式的操作，直接对应该制式的寄存器进行访问。
- 逻辑CPU粒度寄存器：每种制式都有所有逻辑CPU的接口地址，通过逻辑CPU资源虚拟化，将所有逻辑CPU资源虚拟化给不同制式使用，当某种制式访问这些逻辑CPU资源时，通过地址译码出来是特定制式的V_CPU操作，再通过LCPU与V_CPU的关系得到当前的操作对应的逻辑CPU进行访问。
- Group/Queue粒度寄存器：每种制式都有所有Group的接口地址，通过Group资源虚拟化，将所有Group资源虚拟化给不同制式使用，当某种制式访问这些Group资源时，通过地址译码出来是特定制式的V_Group操作，再通过V_Group与P_Group的关系得到当前的操作对应的P_Group进行访问。

3.5.13 任务的缓存

3.5.13.1 新任务缓存

内部缓存空间

ADD_PACKET命令添加的包任务和核间通信消息会缓存到POE中的200个缓存队列和5个FORWARD_TM任务转发队列中。

支持所有内部队列和转发队列共享2560个片内缓存空间（位宽256bit），提供VF最大空间、单队列最大空间限制和最小空间保证。其中最大空间限制是指单队列能够占用的内部缓存空间（包含独占空间和共享空间）上限；最小空间保证是指单队列能够保证独占的内部缓存空间。

VF最大空间按VF配置POE_VF_MAX_NUM。

- 单队列最小空间全局配置POE_Q_MIN_TP（推荐4/8/16/32共4个模板）和POE_Q_MIN_INDEX，按队列配置最小空间保留使能POE_IN_Q_MIN_RSV_EN。
- 单队列最大空间按队列配置，POE提供16个模板POE_Q_MAX_TP，软件只需配置模板索引POE_IN_Q_MAX_INDEX。

所有VF的配置的最大空间之和小于等于2560的空间，同一VF下的所有开启最小保留使能的最小值之和要小于等于该VF的最大空间。

须知

若添加到队列的任务长度大于队列配置的任务长度，POE丢包，上报异常中断，统计丢弃计数。

内部队列共享机制要求为对时延敏感的队列开启最小保留使能，开启后POE保证为该队列预留内部缓存资源和片内溢出缓存资源。

为保证抗突发能力，建议POE队列内部队列最小深度配置为4，且多核共享队列时最大配置为48。

溢出空间缓存。

- 片内溢出空间：



支持所有队列（包含200个缓存队列和5个FORWARD_TM转发任务队列）共享1280个片内溢出缓存空间（位宽256bits），提供VF最大空间、单队列最大空间限制和最小空间保证。

其中VF最大空间、单队列最大空间限制和最小空间保证值均为片内缓存空间的配置值的1/2。所有VF的配置的最大空间之和小于等于1280的空间，同一VF下的所有开启最小保留使能的最小值之和要小于等于该VF的最大空间。

处理单元添加一个任务时，POE将添加的任务缓存到对应内部缓存队列中。内部缓存队列任务占用空间到达队列的最大空间限制，或者VF内共享空间已耗尽时，则添加到片内溢出队列，开启溢出功能时，开始往DDR溢出；不开启溢出功能时，片内溢出空间缓存队列任务到达队列的最大空间限制，或者共享溢出空间已耗尽时，POE不再接受该队列的任务添加。

- **DDR溢出空间：**

DDR的溢出空间按制式进行管理，每个制式单独划分一块连续的DDR空间存放溢出的任务，空间大小必须是16kByte（512个256bit任务）的整数倍，空间大小最小16kByte，最大32MByte（1M个256bit任务）。

这个DDR溢出空间资源是由软件静态分配的，不是POE通过BMU申请的。POE提供5套DDR溢出空间资源给VF0~4使用。

每种制式DDR空间的大小和起始地址可独立寄存器静态配置，每种制式独立初始化DDR溢出空间，必须完成DDR溢出功能初始化后才能进行任务溢出，DDR溢出功能初始需要配置下列寄存器：

- DDR溢出空间首地址通过配置寄存器POE_OVER_BASE_H_ADDR、POE_OVER_BASE_L_ADDR（按制式配置配置）。
- DDR溢出空间大小通过配置寄存器POE_OVER_LEN（配置单位以16kByte为单位，若开启溢出功能，则最小值1，最大值2048）（按制式配置）。
- 提供使能配置寄存器POE_GRP_OVFL_EN和POE_VF_FWD_OVFL_EN，配置每个Group中的任务是否溢出到DDR，0表示禁止该Group的队列溢出，1是允许该Group的队列溢出（按制式配置）。

须知

溢出空间必须静态配置，不能动态配置。

- 若使能了溢出功能，但DDR空间未完成初始化时，不允许开启业务，否则现象不可预知。
- 若回读不及时返回7F任务给CPU，不需要软件重试，POE在包任务回读后会唤醒CPU。

每个制式独立的DDR溢出空间，同一个制式内的所有队列共享DDR空间，每块DDR空间分成若干个16kByte大小的空间，每个队列的溢出空间用链表方式维护16kByte的空间块。如果有队列需要溢出，就向DDR空间申请一块16kByte空间使用，此16kByte空间溢出使用完时，再申请下一块16kByte空间使用。当整个16kByte空间的任务全部回读完成后，这个16kByte空间才被释放，才能被再次申请用作溢出。

当内部缓存队列有空闲存储空间时，从DDR对应队列空间回读任务到内部缓存队列中。

POE提供按制式统计溢出空间资源剩余历史最小值寄存器

HISTORY_FREE_BLOCK_MINNUM，这个统计值是以16kByte为单位配置，即1代表16kByte空间大小，用寄存器POE_OVER_LEN的减去HISTORY_FREE_BLOCK_MINNUM的值，即可得到溢出空间历史最大使用值。



须知

HISTORY_FREE_BLOCK_MINNUM初始值为0x3FFF，在溢出空间未使用时，读该寄存器返回值为0x3FFF，在溢出空间初始化完成后，同步刷新该寄存器为初始值。

任务缓存间的行为描述

处理单元添加一个任务时，POE首先将添加的任务缓存到对应的片内溢出空间缓存队列中，然后根据不同情况进行下一步处理：

- 场景1：若此时该队列的DDR溢出空间中没有任何任务，则将该任务转移到内部缓存空间中；当内部缓存队列任务占用空间到达队列的最大空间限制，或者共享空间已耗尽时，则任务仍然暂存在片内溢出空间中，在开启溢出功能时，开始往DDR空间溢出；不开启溢出功能时，片内溢出空间缓存队列任务到达队列的最大空间限制，或者共享溢出空间已耗尽时，POE不再接受该队列的任务添加。
- 场景2：若此时该队列的DDR溢出空间中有任务，为了保证任务顺序，则将任务直接转移到DDR溢出空间；当对应队列的内部缓存空间有足够资源时，POE从DDR溢出空间将任务回读至内部缓存空间。

处理单元获取的任务均从内部缓存空间获取。

任务溢出回读行为描述

- 一个任务被加入POE，POE内部逻辑判断该数据所属队列是否在溢出空间（包含POE内部的溢出缓存和外部DDR的溢出缓存）已有数据存储，若没有，则存入POE内部该队列非溢出缓存空间，否则就存储到POE内部的溢出缓存，再由POE内部逻辑操作将POE内部溢出缓存数据溢出到DDR或溢出回POE内部该队列非溢出缓存空间（溢出需要遵循队列内数据保序原则）。这里注意对于每一个队列来说，需要溢出的数据凑齐一个cachline，POE才会为此队列发起一次溢出操作。
- POE的一个队列在POE内部缓存空闲空间大于等于一个64Byte时，若此时DDR中有该队列数据，则进行回读操作，一次回读一个64Byte的数据。
- 处理单元获取任务时，都是从对应队列的POE内部缓存来获取，如果内部缓存无有效数据，则POE给处理单元返回无效数据。

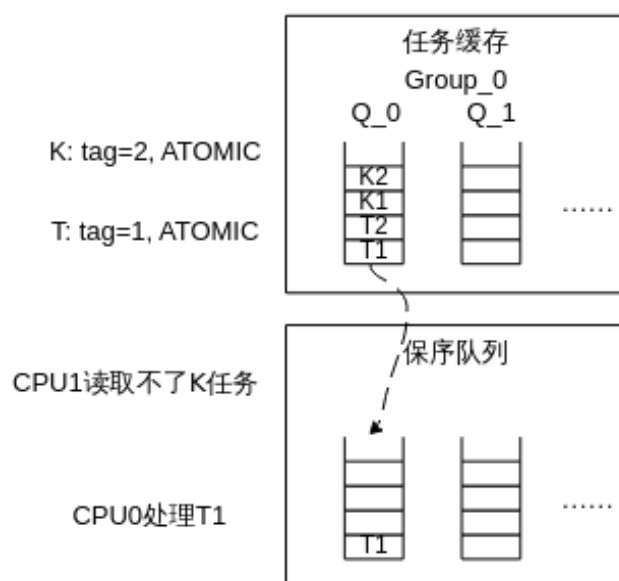
须知

当前POE中，所有队列共享溢出空间，一个数据块在读空释放后才能被再次使用；如果在数据块此时溢出满，后续又回读了一半空间，现在再添加任务需要溢出时，数据块也只剩一半的空间可以用做溢出，所以数据块会有利用率的问题。在边界条件下，数据块会有利用不充分的情况，最大任务误差“数据块任务 - 1”。

3.5.13.2 Atomic 流拥塞

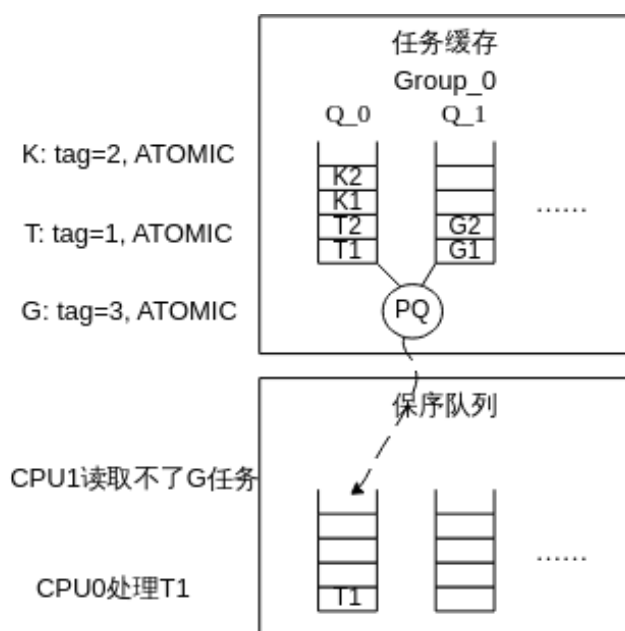
流拥塞问题和调度拥塞问题如下：

图 3-3 流阻塞问题



在逻辑CPU0、CPU1同时绑定Group_0，且Group_0配置为保序，且所有任务都添加到队列0（Q_0）中，所有任务都是ATOMIC类型的包任务，参考上[图3-3](#)场景。根据ATOMIC类型包任务的特性，在逻辑CPU0读取T1包任务在处理时，逻辑CPU1不能读取相同流的T2包任务处理，但T2在队列头，阻塞K流的任务，逻辑CPU1也无法获取K流任务处理。

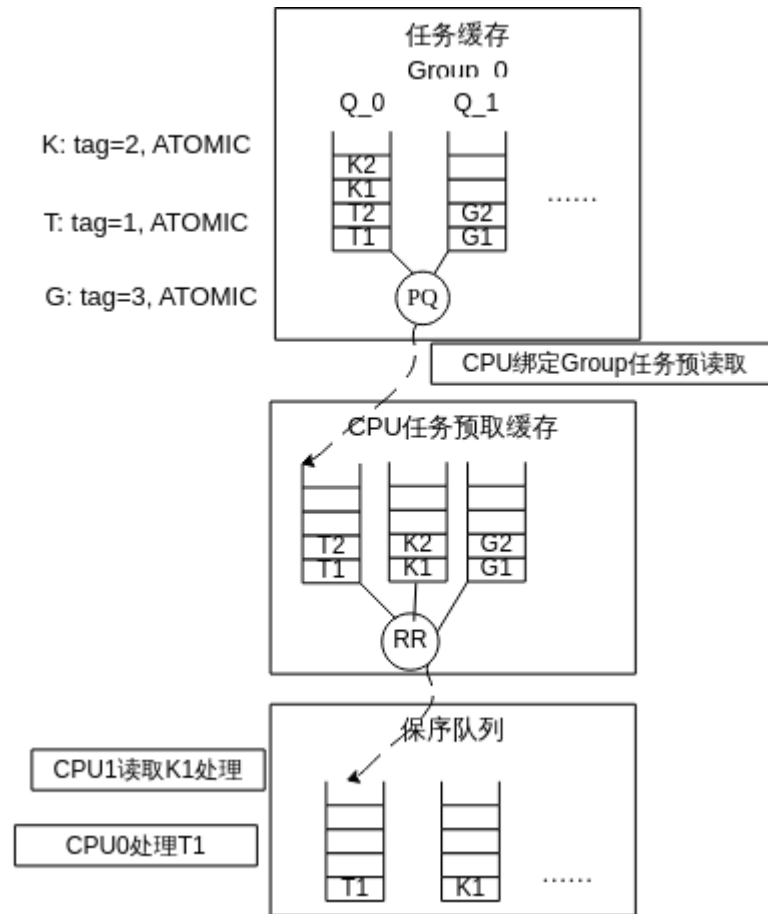
图 3-4 调度阻塞问题



在逻辑CPU0、CPU1同时绑定Group_0，且Group_0配置为保序，且Group_0中队列配置为PQ（绝对优先级）调度，且队列中所有任务都是ATOMIC类型的包任务，参考[图3-4](#)场景。根据ATOMIC类型包任务的特性，在逻辑CPU0读取T1包任务在处理时，逻辑CPU1无法获取G流任务处理。

辑CPU1不能读取相同流的T2包任务处理，但同时Q_0和Q_1间是PQ调度，也无法调度G流的包任务进行处理。

图 3-5 CPU 任务预取缓存



须知

为缓解Atomic流拥塞调度问题，流分类GROUP0~3的流分类队列支持Atomic流禁止调度功能，在流分类队列内有某个Atomic流的任务已完成保序注册，此时会禁止该Atomic流的其他任务再次调度，但不影响其他流的调度，直到该流消亡后才会解除禁止调度，非流分类的其他GROUP不支持Atomic流禁止调度功能，由软件解决Atomic流拥塞调度问题。

流分类队列的禁止调度会将Atomic Parallel和Atomic Serial任务悲观的等同Atomic Serial处理，不支持并行功能。

3.5.13.3 CPU 解绑定任务缓存

一个保序包任务在被一个VCPU处理时，是不能同时被其他VCPU处理的。同样，一个VCPU在处理一个保序包任务时，是不能同时处理另一个保序包任务的（PUSH_LCPU例外）。

当一个保序包由一个VCPU处理时，POE会标记该VCPU已经有一个保序包任务在处理。当该VCPU发送SWITCH NULL、SWITCH DESCHEDULE、DESCHEDULE以及



SEND_TM/ POE命令时，POE会将该VCPU绑定状态解除，并且如果是SWITCH DESCHEDULE和DESCHEDULE这两个命令，对应这个VCPU处理的包任务会进入POE内部的解绑定任务缓存，使逻辑CPU可获取处理。

解绑定任务缓存是由16个单独的缓存队列构成，对应VCPU能处理的所有物理Group，每个物理Group有一个缓存队列。这16个缓存队列共用128个任务描述符缓存。

VCPU发送GET_POE和GET_GROUP优先从解绑定任务缓存中获取任务。获取到的保序包，POE会标记该VCPU已经有一个保序包任务在处理，该保序包任务信息在第一次被VCPU获取时，已经进入POE的保序队列进行保序，解绑定（CPU发DESCHEDULE或SWITCH_DESCHEDULE命令）后，该保序包依然在POE的保序队列中进行保序。

3.5.13.4 CPU 流分类任务缓存

POE仅支持为Group0~Group3提供流分类任务调度，支持Group0~Group3的流分类调度使能通过POE_GROUP_AFFINITY_EN配置；分别提供一个深度为32的流分类任务缓存队列，共4个流分类任务缓存队列。

- 流分类任务缓存：

当Group0~Group3的流分类使能打开时，对Group0~Group3的内部缓存中的任务分别按照GROUP内PQ+WRR调度算法进行读取，读取的任务放入对应流分类任务缓存队列。

为保证任务调度的QoS特性，基于Group内队列QoS共享32个任务缓存深度，其中为每个流分类队列QoS7~QoS0分别提供一个占用最大独占一个任务缓存的门限配置寄存器，并参与共享32个任务缓存。当共享任务缓存未耗尽或者且当Group某个QoS任务个数小于其门限值为0时，可以从该Group可以读取该QoS的队列任务并缓存到流分类队列，否则不能进行读取。

- 流分类任务调度：

VCPU发GET_POE和GET_GROUP获取任务指令时，流分类队列首先同内部缓存队列一同参与PQ+RR+WRR/RR+PQ+WRR调度，调度出一个GROUP，若调度出的GROUP为流分类GROUP，则按照流分类调度算法调度出一个包任务，若调度出的任务是包任务且配置为保序，包信息入POE保序队列，并且包任务和CPU建立绑定关系。

须知

VCPU GET_QUEUE固定不支持从流分类任务缓存队列获取任务，在Debug场景下，可以使用GET_GROUP方式将流分类任务缓存队列中的任务取出。

若动态调整同队列的QoS，POE不保证调整后一段时间内的同队列出包顺序，若要避免此场景需要软件在动态调整前排空队列中的任务。

由于流分类队列的QoS门限机制，在流分类队列内的高优先级的包任务出现调度冲突，但是低优先级的任务由于门限限制导致无法预取到流分类队列内，可能导致低优先级业务断流，该现象符合设计和产品预期。

3.5.14 加速器任务处理通知

POE模块提供任务处理通知功能触发加速器到POE获取任务，POE会向加速器发送消息以通知各个加速器来取包。

POE通过PF统一发送消息给各加速器的PF，发送该消息的地址由POE中PF制式的寄存器POE_MSG_ACC_PF_ADDR_H_CFG、POE_MSG_ACC_PF_ADDR_L_CFG。POE添加



各加速器的任务通知命令的Outstanding个数可通过寄存器POE_SEND_MSG_OTD_NUM配置。

POE给加速器发送任务处理通知命令的触发条件：

- POE中一个循环计时器，达到定时寄存器POE_MSG_TMOUT_CFG 配置值（该门限值由PF配置，不分制式）时，给各个加速器遍历发送一次任务通知。
- POE中加速器可GET_POE的Group任务从无到有和从有到无时，发送任务通知命令更新加速器中任务通知寄存器的任务状态。

须知

当启用一个加速器的一个制式（VF0~4）业务功能时，必须保证该加速器的PF制式已经能够接收来自于POE的任务通知消息的写寄存器操作时，才可通过配置POE的相应VF0~4下的POE_GRP_ACC_MASK和POE_MSG_ACC_EN对应bit位为1，其中POE_MSG_ACC_EN按VF可配置加速器使能。

3.5.15 任务负载均衡

3.5.15.1 Group 的划分

POE给逻辑CPU提供了16*8（共16个物理Group，每个Group有8个队列）的物理Group资源，逻辑CPU分配的物理Group资源编号固定从0~15；POE给加速器提供了18*4（共18个物理Group，每个Group有4个队列）的物理Group资源，加速器分配的物理Group资源编号固定从16~33。

须知

- 软件使用用于与CPU绑定的虚拟Group时，这些Group编号必须从0开始；
- 软件使用用于与加速器绑定的虚拟Group时，这些Group编号必须从16开始。

3.5.15.2 负载均衡配置

任务负载功能通过物理Group划分实现，每个物理Group中的任务可以分配一个或多个处理单元进行处理。

POE提供一组VCPU和加速器的V_Group均衡配置寄存器。每个VCPU在POE中有一组配置寄存器POE_GRP_VCPU_MASK，每个能从POE获取任务的加速器都有一组配置寄存器POE_GRP_ACC_MASK，用来配置VCPU和加速器可以处理哪些V_Group。

配置以上寄存器的每1bit代表对应的一个V_Group，将对应bit设置为1，表示该处理单元可以处理对应的V_Group中的任务。寄存器默认全0，表示处理单元不处理V_Group的任务。

在配置了均衡配置寄存器后，可以得到VCPU和加速器可以处理的物理Group。



表 3-3 处理单元和 Group 映射关系配置举例

配置寄存器名称	配置值	含义
GRP_VCPU_MASK_0	0x3	VCPU_0可以处理 V_Group_0和V_Group_1 的任务。
GRP_VCPU_MASK_1	0x4	VCPU_1可以处理 V_Group_3的任务。
GRP_ACC_MASK_0	0x7	ACC0可以处理 V_Group16、V_Group17 和V_Group18的任务。
GRP_ACC_MASK_1	0x8	ACC1可以处理 V_Group19的任务。

通过上面配置也可以将POE作为相同制式中的路由使用，比如处理单元0将处理完的任务添加到Group1中，处理单元1从Group1中获取任务进行处理，这样就实现了处理单元0到处理单元1的任务传递。

配置MASK寄存器后，查询WR_CPU_MASK_FAIL和WR_ACC_MASK_FAIL寄存器确认是否有配置MASK失败的状态，若有失败状态，软件可采取以下步骤重新配置MASK寄存器。

对于配置VF的CPU的MASK寄存器：

- 步骤1
- 读取WR_CPU_MASK_UNEVEN_CNT寄存器，若读该寄存器返回值为0，软件可以重新配置MASK寄存器；若查看该寄存器的返回数值n（0<n<M），则需要进行以下步骤。
- 步骤2
- 写M-n次VCPU的POE_GRP_VCPU_MASK（属于无效配置，仅为了抵消之前配置次数小于M引入的配置错误）。
- 步骤3
- 读取WR_CPU_MASK_UNEVEN_CNT寄存器，若读该寄存器返回值为0，软件可以重新配置MASK寄存器。若WR_CPU_MASK_UNEVEN_CNT返回值n（0<n<M），重复步骤2。

----结束

须知

M=3，需要连续正确写3次mask寄存器。

对于配置VF的加速器的mask寄存器：

- 步骤1
- 读取WR_ACC_MASK_UNEVEN_ST寄存器，若读该寄存器返回值为0，软件可以重新配置MASK寄存器；若读该寄存器返回值为1，则需要进行以下步骤。
- 步骤2
- 写一次ACC的POE_GRP_ACC_MASK（属于无效配置，仅为了抵消之前的奇数次配置引入的错误）。



步骤3 读取WR_ACC_MASK_UNEVEN_ST寄存器，若读该寄存器返回值为0，软件可以重新配置MASK寄存器；若读该寄存器返回值为1，重复**步骤2**。

----结束

须知

当同一个Group同时被划分给不同硬件加速器时，会造成功能紊乱，POE硬件逻辑不做这个配置冲突检查。

POE模块VCPU和加速器的Group掩码寄存器，同制式下同一个处理单元的MASK寄存器需要连续配置，并且只有当同一个处理单元的MASK寄存器都配置完成后，该组的MASK寄存器配置才生效。若在同一处理单元的MASK寄存器中间插入其他MASK寄存器的写操作，会导致POE中Group掩码寄存器的配置错误。

3.5.16 任务 QoS 调度管理

POE中队列支持8个优先级，优先级由高到低是QoS7>QoS6>...>QoS0。每个制式都有对应的寄存器POE_Q_SCH_QOS_CFG来配置每个队列的优先级。

3.5.16.1 逻辑 CPU 的任务调度

在属于逻辑CPU的P_Group中有任务时，POE就会通知逻辑CPU来POE进行任务获取操作。逻辑CPU使用GET_POE/GET_GROUP命令进行任务获取时，并不会指定队列，而是由POE模块按照特定调度策略来进行任务调度，选择一个任务返回给该逻辑CPU。

- POE提供寄存器POE_Q_SCH_QOS_CFG配置队列优先级，按制式进行配置，每种制式只配置制式内的队列号对应的优先级。
- POE提供寄存器POE_Q_WRR_CFG配置队列权重，按制式进行配置，每种制式只配置制式内的队列号对应的权重，权重值为0~255。
- POE提供PQ+RR+WRR的三级任务调度方式，逻辑CPU或加速器发起GET_POE命令获取任务时，流分类队列和内部缓存队列一起参与QoS调度：QoS间采用PQ调度，同QoS采用RR调度。POE首先按照QoS值对绑定的所有队列做PQ调度；然后对最高QoS的队列所属Group做RR调度挑选出一个Group；最后从一个Group中以WRR方式把该QoS的队列调度出POE。

须知

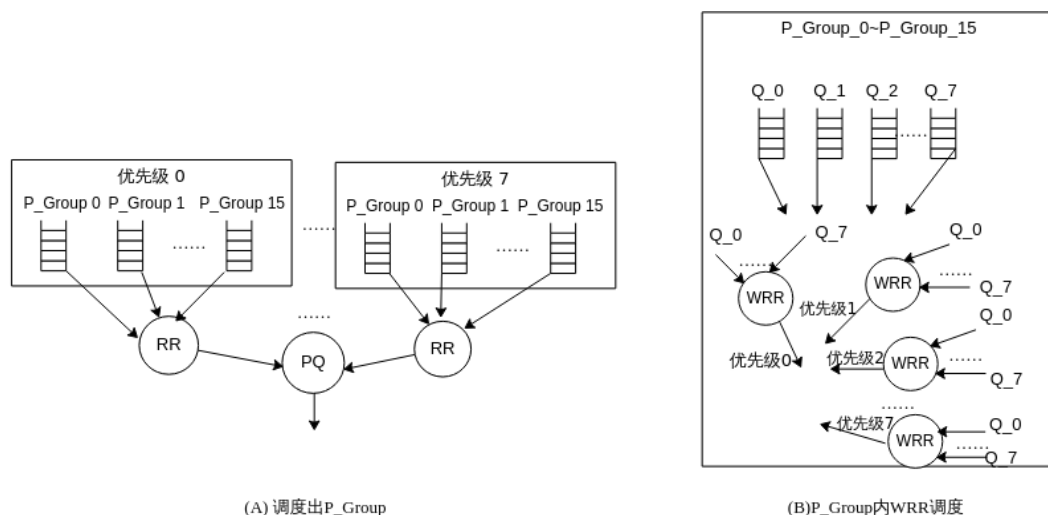
WRR权重配置为0时，该Queue就不会有包被调度出来，调度结果不可预知。

逻辑CPU的任务调度如下：

- **图3-6**中Q_*表示P_Group内部的队列编号，P_Group内部的队列可以根据需要配置为0~7中8个优先级的任意优先级，第一级先进行队列间的优先级PQ（基于QoS的绝对优先级）调度（**图3-6**中A），第一级调度先挑选出该处理单元对应的非空缓存队列中QoS最大的所有缓存队列。
- 第二级进行P_Group的RR调度（**图3-6**中A）：先得到第一级调度挑选出来的队列所属的所有P_Group，将这些P_Group进行RR调度，挑选出一个P_Group。

- 第三级进行P_Group内缓存队列的WRR调度（图3-6中B）：第二级调度得到挑选出的P_Group之后，将该P_Group中所有第一级PQ调度挑选出的缓存队列进行WRR调度，得到一个缓存队列。

图 3-6 逻辑 CPU 任务调度图



POE提供RR+PQ+WRR的三级任务调度方式，逻辑CPU发起GET_POE命令获取任务时，POE首先对绑定的流分类队列和内部缓存队列做RR调度挑选出一个Group；然后对该Group内最高QoS的所有队列做WRR调度挑选出一个队列。

逻辑CPU的任务调度如下：

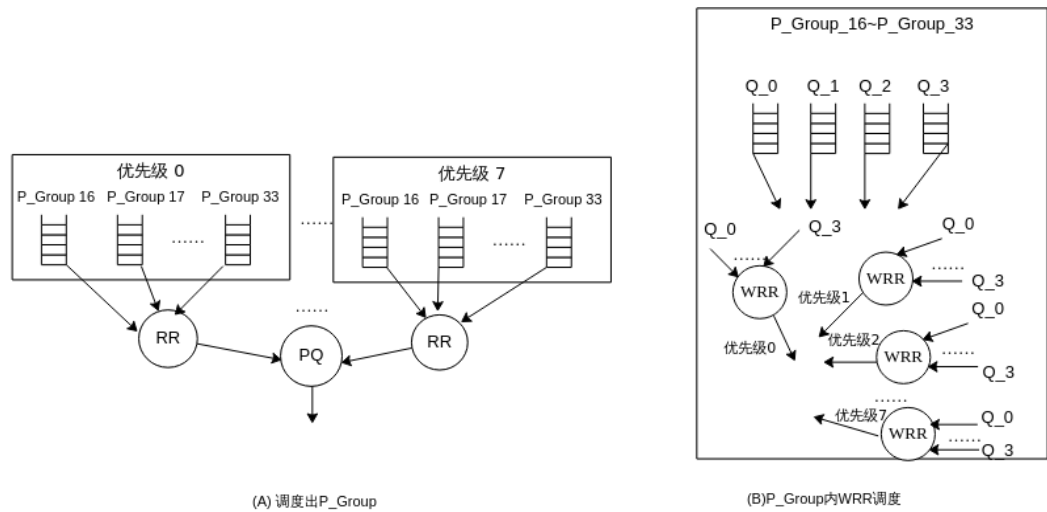
- 第一级进行P_Group之间的RR调度，挑选出一个P_Group。
- 第二级对上一级挑选出的P_Group内部队列间进行优先级PQ（基于QoS的绝对优先级）调度，挑选出所有QoS最大的所有缓存队列。
- 第三级对上一级挑选出的QoS最大的缓存队列进行WRR调度，得到一个缓存队列。

3.5.16.2 加速器的任务调度

POE中加速器的P_Group中有任务时，POE就会通知加速器来POE进行任务获取操作。加速器固定使用GET_POE命令进行任务获取，并不会指定队列，而是由POE模块按照特定调度策略进行任务调度，按照PQ+RR+WRR的三级任务调度方式，选择一个任务作为加速器获取任务操作的响应数据返回给该加速器。

加速器的任务调度如下图3-7所示。

图 3-7 加速器的任务调度图

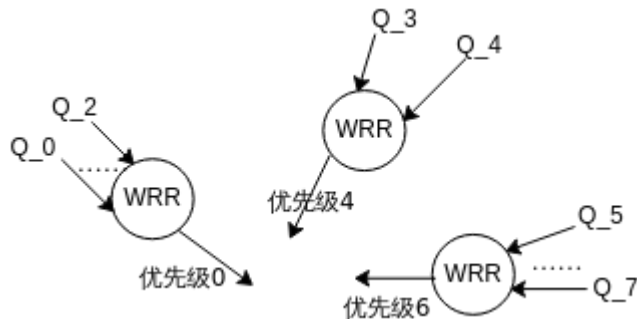


3.5.16.3 调度场景说明

下面举例说明，详见[案例1](#)和[案例2](#)（可用于主控传输场景）。

案例 1

图 3-8 例 1 Group 内优先级配置

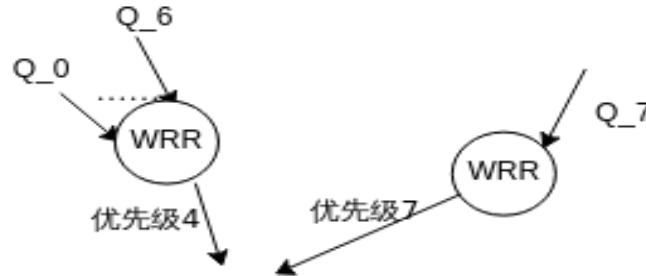


比如3个Group（Group0，Group1，Group2）内部都配置为3个优先级：如[图3-8](#)所示。

那么进行第一级PQ调度时，只有QoS0、QoS4、QoS6三个优先级参与调度，QoS6获得仲裁权；第二级RR调度从3个Group选择一个Group，比如选择Group2；第三级WRR调度，对Group2内QoS6的WRR调度，最终从Q_5/Q_6/Q_7中选择一个队列进行任务出队。

案例 2（可用于主控传输场景）

图 3-9 例 2 Group 内优先级配置



3个Group（Group0，Group1，Group2）内部的8个队列都配置为两个优先级：如图3-9所示。

那么进行第一级PQ调度时，只有QoS7、QoS4两个优先级参与调度，QoS7获得仲裁权；第二级RR调度从3个Group选择一个Group，比如选择Group2；第三级WRR调度，从Group2内QoS7的Q_7队列进行任务出队。

如果Q_7中无任务，第一级PQ调度时，只有QoS4一个优先级参与调度，QoS4获得仲裁权；第二级RR调度从3个Group选择一个Group，比如选择Group2；第三级调度从Group2内QoS4的WRR调度，最终从Q_0～Q_6中选择一个队列进行任务出队。

3.5.17 包任务保序（多核并发处理同流）

POE只为VCPU处理的包任务进行保序，包任务依照：TAG+OG+TT进行保序（拥有相同TAG、OG、TT域值的包任务，定义为同一个流）。其中TAG和TT为包任务数据结构携带，OG通过POE_GRP_ORDER_TG配置。由于OG是VF内部的独立编号，所以保序时同一条流需要加上VFID，同时，TT=ATOMIC Serial和TT=ATOMIC Parallel，这两种TT认为是同一个TT。

VCPU处理的V_Group的包任务是否需要保序，由相应制式的GRP_ORDER_FLOW_EN（配置各制式中使用的V_Group是否需要保序）寄存器和POE_GRP_VP_MAP、POE_GRP_VF_MAP（配置P_Group与各制式的V_Group的映射关系）寄存器设定。对于各个加速器处理单元，这些加速器发起GET_POE指令时，POE不会给这些包任务进行保序。

须知

在POE中，只有逻辑CPU处理的包任务Group可设置为保序；其中核间消息固定不保序（即使对应Group设定为保序）；加速器处理的包任务Group，固定不保序。

逻辑CPU获取任务后，需判断此任务是否在POE中保序，如保序，必须发送保序命令和POE交互，否则该逻辑CPU不能再向POE获取到任务；如不保序，逻辑CPU不必发送保序命令给POE。

在VCPU获取包任务后，如果此包任务需要保序，包信息进入POE保序队列，并且包任务和该VCPU建立绑定关系；如果此包任务不需要保序，包任务信息不进入POE保序队列，POE中也不再保留包任务信息。VCPU可通过查询对应制式的POE_LCPU_BIND_INFO [0]来确认该VCPU是否与VCPU正在处理的包任务绑定。



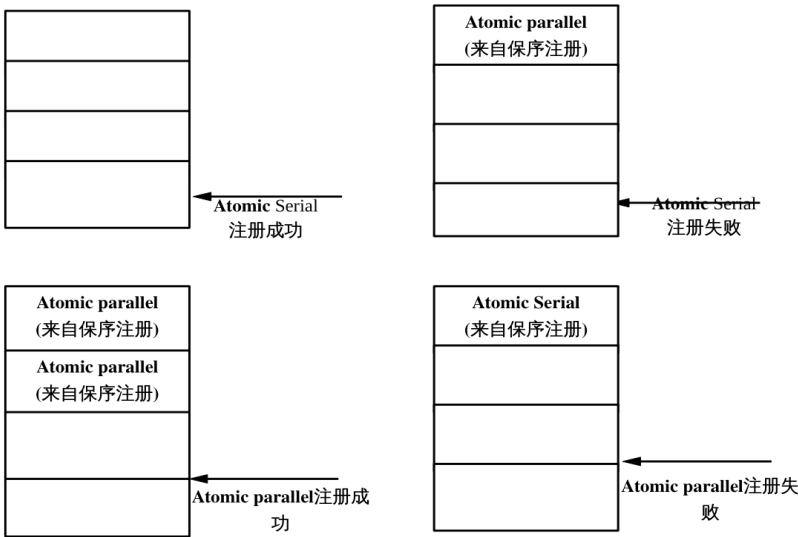
POE内部保序队列最多可以保存128个保序包任务信息，最多支持到128个保序队列。

针对ATOMIC任务，支持基于集合MASK的互斥调度，支持基于MASK进行两个TAG是否互斥的比对；支持对需要互斥的TAG进行互斥调度（并行执行和串行执行）。Hi309a中只支持如下互斥调度，Tag_A 和Tag_B两个Tag值的ATOMIC任务，且ATOMIC任务的VF和OG相同，若 $(Tag_A \mid Tag_B) == Tag_A$ 或 $(Tag_A \mid Tag_B) == Tag_B$ 时，认为二者互斥，只能串行执行；否则，二者可并行执行，通过配置POE_ATOMIC_MASK_EN可以开启和关闭集合MASK互斥调度功能。

针对ATOMIC任务，根据两种不同的情况分别说明如下：

- 关闭基于集合MASK的互斥调度的保序。
 - ATOMIC Serial任务保序注册时，检查目标流是否为空（可能由于保序注册变为非空）。目标流队列为空时注册成功，目标流队列非空时注册失败。
 - ATOMIC Parallel任务保序注册时，检查目标流是否存在保序注册进入的ATOMIC Serial包。当存在保序注册进入的ATOMIC Serial包时，注册失败；反之，则注册成功。
- 如图3-10所示。

图 3-10 关闭基于集合 MASK 的互斥调度 ATOMIC 流注册示意图



- 使能基于集合MASK的互斥调度的保序

图 3-11 使能基于集合 MASK 的互斥调度的 ATOMIC 流注册示意图

互斥的前提：在TagA的流已经注册的情况下，对 ATOMIC流，同时除TAG外，TG和VF相同

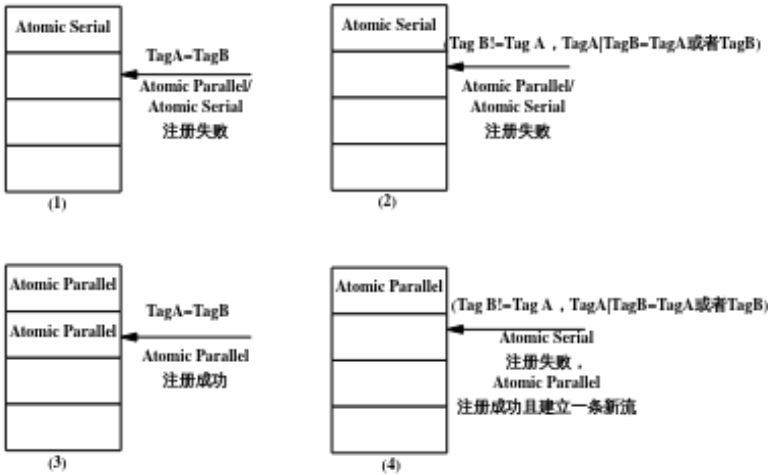


图3-11中的（1）、（3）的情况与图3-10没有差别。唯一的差别在（2）和（4）的场景。

- 在（2）的场景中，在关闭基于集合MASK互斥调度时ATOMIC Parallel或ATOMIC Serial的TagB任务会注册到一条新的流中，但是现在TagB的任务不能注册成功。
- 在（4）的场景中，在关闭基于集合MASK互斥调度时ATOMIC Parallel或ATOMIC Serial的TagB任务会注册到一条新的流中，但是现在属于ATOMIC Serial的TagB任务不能注册成功，ATOMIC Parallel的TagB任务依然能注册成功。

须知

- 基于集合MASK互斥调度与流分类功能互斥，同一个GROUP内二者不能同时开启。
- 基于集合MASK互斥调度与跨GROUP保序功能互斥，同一个GROUP内二者不能同时开启。

对于推送cacheable私有队列，私有队列深度为8，则需要支持每个LCPU同时绑定9个保序任务，因此需要在POE内部为每个LCPU提供一个10深的绑定信息队列，已注册成功的保序包，建立保序包任务与绑定单元的绑定关系，同时在任务数据结构中增加“Order_BindID”域段，当CPU完成包处理后，需要发送保序命令清除指定绑定单元的绑定关系并交由POE完成下一阶段处理。

在POE中为每个LCPU提供10个绑定单元，最多支持LCPU同时绑定10个保序任务，其中兼容模式GET_POE/GET_GROUP固定使用每个LCPU的绑定单元0，推送私有队列固定使用绑定单元1~9。

3.5.17.1 保序队列

POE保序队列采用链式存储方式实现。

VCPU通过GET_POE/GET_GROUP命令获取POE中的包任务或POE推送任务到LCPU私有队列时，如果该任务是需要保序的，那么这个包任务就会添加到保序队列中进行保序管理。



保序队列的划分是按照VF、OG、TT、Tag这四个域值。这四个域值都相同的包任务被POE安排划分入同一个保序队列。

保序单元资源是所有制式共用的，POE为每种制式提供保序单元水线配置寄存器POE_INFLIGHT_TH，通过配置各制式的保序单元资源水线，确保不会存在某种制式因为其他制式将保序单元资源用完而没有保序单元资源使用。当保序单元使用个数到达保序单元水线时，该制式的VCPU不能在POE获取到保序Group中的包任务。当某制式下的保序空间使用达到水线，或者整体保序空间已经都被使用的情况下，CPU获取无效（SourcePort=0x7F）任务。

须知

同一制式的CPU背靠背的通过GET_POE或GET_GROUP命令获取保序任务时，该制式的保序任务个数最多要到达保序水线寄存器POE_INFLIGHT_TH配置值再加4时，保序水线反压才生效。

3.5.17.2 保序命令处理

保序命令中SWITCH_NULL、SWITCH_DESCHEDULE、DESCHEDULE、SEND_TM、SEND_POE，在保序任务添加到POE后，任务与当前逻辑CPU解绑定。

SWITCH_NULL 命令

图 3-12 SWITCH_NULL 命令执行

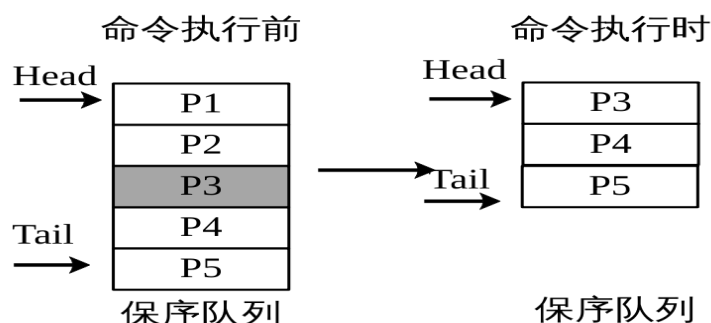
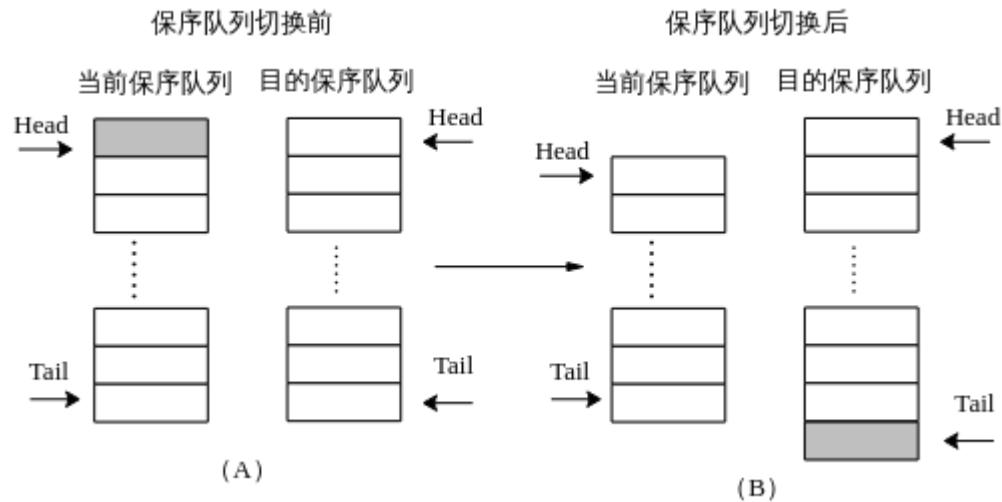


图3-12中针对P3包任务执行SWITCH_NULL命令，当包任务到达当前保序队列的队头时，从保序队列中删除。

SWITCH_DESCHEDULE 命令

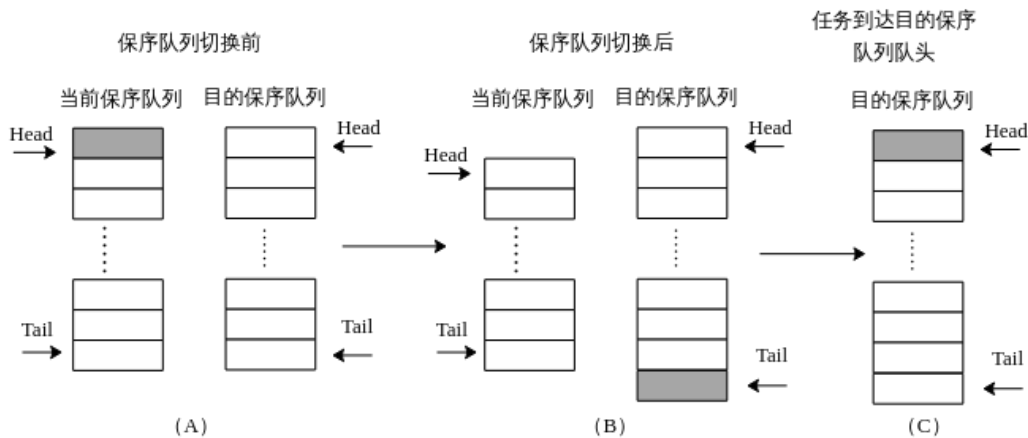
命令的执行动作因TT的不同而不同。

图 3-13 TT 为 ORDER 的 SWITCH_DESCCHEDULE 命令执行



TT是ORDER时，当包任务到达当前保序队列的队头时，才能向目的保序队列进行切换（如上[图3-13A](#)），当包任务从当前保序队列切换到目的保序队列后（如上[图3-13B](#)），SWITCH_DESCCHEDULE命令就算完成，包任务放入“解绑定缓存队列”中，包任务可以进行再调度，被其他逻辑CPU获取。

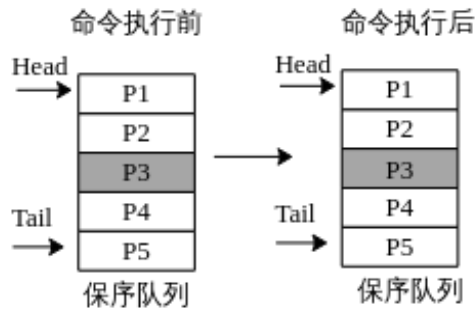
图 3-14 TT 为 ATOMIC/NULL 时的 SWITCH_DESCCHEDULE 命令执行



Nxt_Order_Type是ATOMIC/NULL时，SWITCH_DESCCHEDULE命令等同于SWITCH_NULL命令。

DESCCHEDULE 命令

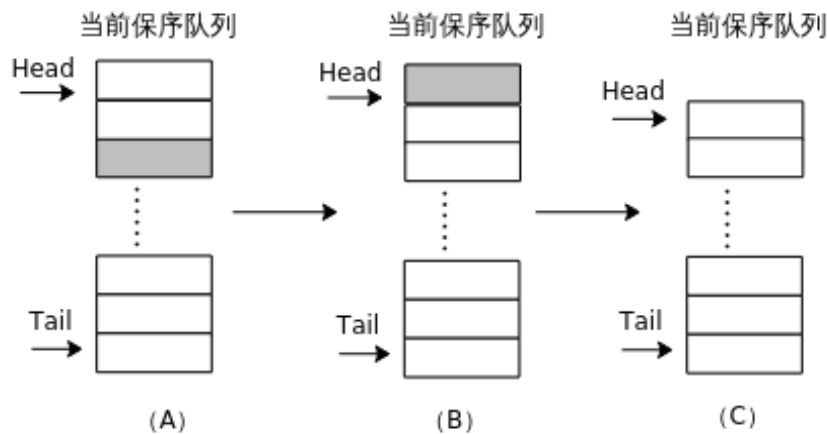
图 3-15 DESCCHEDULE 命令执行



包任务的保序信息在保序队列中的位置不变，包任务放入“解绑定缓存队列”中，包任务可以参与再调度，可再次被逻辑CPU获取处理。

SEND_TM、SEND_POE 命令

图 3-16 SEND_TM、SEND_POE 命令执行



当包任务到达当前保序队列的队头时（如上图3-16B），才能将包任务从保序队列中删除保序信息（如上图3-16C），包任务被推送给下一个模块，当包任务被推送到目的模块后，SEND_TM、SEND_POE命令才算执行完毕。

保序队列队头超时

保序队列中，如果队头任务长时间没有处理单元来处理，上报中断给VF，由VF软件通过绑定的VCPU发送DESCCHEDULE命令将队头任务放到解绑定缓存中，供其他VCPU获取处理。

保序队头超时中断对应中断状态寄存器位是POE_INT_STATUS.flight_tout_int。中断信息寄存器是POE_FLIGHT_TOUT_INT_INFO_0，记录该超时的保序队列队头使用的保序EntryID信息的信息。并且，软件可以通过查询对应制式的POE_INFLIGHT_TH_INT_INFO寄存器来获知POE中对应制式当前使用的保序单元个数。



3.5.18 包任务通讯

处理单元间可以通过POE以包任务的形式进行任务交互，处理单元1的一个VF向BMU中对应VF申请一个Buffer，将任务数据写入Buffer中，生成一个任务描述符，将任务描述符写入POE中处理单元2的对应VF能处理的V_Group队列中；处理单元2的VF从该V_Group队列中获取此任务描述符，通过任务描述符中的地址，从Buffer中读出任务数据进行处理。

包通讯用到的命令：ADD_PACKET、GET_POE、GET_GROUP、GET_QUEUE、SWITCH_NULL、SWITCH_DESCCHEDULE、DESCCHEDULE、SEND_TM、SEND_POE。

处理单元每个VF在向POE加包时，其Group编号都是该制式下的V_Group编号，另一个处理单元使用GET_POE获取包时，只能从POE中得到该制式下V_Group中的包任务。

3.5.19 核间消息通讯

VCPU之间进行直接数据通讯，最大通讯数据是256bit。逻辑CPU0直接将256bits数据写入POE的V_Group队列，逻辑CPU1从对应的V_Group队列中获取此256bits数据，根据软件定义的通讯数据格式进行处理。

核间通讯用到的命令：ADD_SPE_WORK、GET_POE、GET_GROUP、GET_QUEUE。

3.5.20 WFI/WFE 唤醒 CPU 读取任务

等待中断是ARMv8及以上架构中使CPU Core进入低功耗模式的一个特性，在此模式下CPU内部大部分的时钟被关闭，仅保持逻辑处于供电状态，寄存器内容保持，只有一小部分逻辑处于工作状态，将功耗减小到静态漏电。将CPU Core唤醒方式有WFI（中断触发方式唤醒）或WFE（事件触发方式唤醒）模式。

3.5.20.1 事件触发通知（WFE）

POE支持为CPU产生对应的唤醒Event硬件信号线。Event触发信号可将对应物理CPU从睡眠状态唤醒，让CPU从POE读取任务，从而缩短任务处理延时。

POE提供8根Event触发信号，可唤醒8个物理CPU。通过寄存器

POE发送一个高脉冲Event信号给一个物理CPU，取决于两个条件，这两个条件缺一不可：

- POE认为该物理CPU已经进入睡眠状态。
- POE中与该物理CPU绑定的Group中有任务。

在满足唤醒条件的物理CPU中，属于相同制式的物理CPU逐个唤醒，逐个唤醒的时间间隔由寄存器POE_CPU_WFE_TMOUT_CFG配置。

上述两个条件中，针对条件1，POE有两种判断物理CPU是否进入睡眠的方法：

- 方法1：软件通知POE物理CPU是否进入睡眠状态。
软件通过配置POE寄存器POE_CPU_CORE_EVENT_CFG来通知POE物理CPU是否进入睡眠状态。
POE_CPU_CORE_EVENT_CFG值为1，POE将POE_CPU_CORE_EVENT_*_DFX值置1，POE认为该寄存器对应的物理CPU进入了睡眠状态；该寄存器值为0，POE认为物理CPU没有在睡眠状态。
当POE发送Event脉冲信号给物理CPU的同时，POE将寄存器POE_CPU_CORE_EVENT_*_DFX值清零，即POE发送Event脉冲信号给物理CPU时，就认为该CPU已经不在睡眠状态了。



- 方法2：物理CPU从POE读取任务失败后，POE记录状态，且认为对应物理CPU进入睡眠状态。
软件可以通过配置POE的寄存器POE_CPU_WFE_MODE_CFG来让POE使用方法1或方法2判断某个物理CPU是否进入睡眠状态。
因每个物理CPU有一根WFE状态指示信号，那么16个逻辑CPU和8个物理CPU之间需要有个映射关系，以判断唤醒哪个物理CPU。映射关系通过16个映射寄存器POE_LCPU_MAP{0~15}配置，配置逻辑CPU属于哪个物理CPU。

须知

在推送Cacheable私有队列场景下，POE推送Cacheable队列时，固定不推送空调度的7F任务，因此在开启推送Cacheable存储队列时，若物理CPU进入睡眠。需要软件通知POE物理CPU已进入睡眠，POE在有包任务时会唤醒物理CPU。若LCPU的Cacheable私有队列空，POE在推送任务到LCPU的Cacheable私有队列后，会自行WFE唤醒物理CPU；若LCPU的Cacheable私有队列非空，POE在满足轮询条件（POE可推送credit为0）时多次读取Cacheable私有队列的读指针时，如果发现LCPU长时间未从Cacheable私有队列取任务，则POE在超时后自行唤醒CPU，该功能通过POE_PUSH_CACHEQ_WFE_TMOUT_CFG寄存器可配置使能和超时门限。

无论采用软件通知方法还是获取包失败方法，CPU都会通过WFE硬连线通知POE处于sleep状态，只有在硬连线状态为高时，POE根据当前状态选择是否发起WFE唤醒CPU。

3.5.20.2 中断通知（WFI）

每个逻辑CPU对应一个中断线，对应16个逻辑CPU有16个中断进行任务上报，当POE中有物理Group水线中断、物理Group未取包超时中断时，提供中断通知逻辑CPU，当有多个逻辑CPU对应一个物理Group时，此物理Group有任务时，上报中断给该物理Group绑定的逻辑CPU。

3.5.21 多模隔离

Hi309a支持多种无线接入制式（RAT）同时运行，各个制式之间的业务处理需要隔离，采用基于虚拟化的机制来实现多模的隔离。

POE模块对各制式间的多模隔离内容包括寄存器多模隔离、包任务/消息多模隔离、保序命令隔离。

- 寄存器多模隔离：
POE通过PF/VF的方式进行功能及寄存器接口划分，POE支持1个PF和5个VF；PF管理所有VF的公共功能以及各个VF之间的资源分配。VF是能够独立完成业务处理的功能（如与队列相关的配置寄存器，运行过程需要频繁操作的接口寄存器等）。



表 3-4 POE 寄存器划分多模列表

VF/ PF 制式	模块寄存器地址接口	说明	备注
PF	模块级	模块头信息	寄存器地址都有实体寄存器与之对应。
		模块级公共中断	
		模块级公共关键状态	
		模块级配置寄存器	
		模块级公共维测统计	
	PF_VF0	PF访问该VF的中断	1. CPU、GROUP相关的寄存器只有地址，需要映射到“业务级_PF”实体寄存器； 2. 其余寄存器地址都有实体寄存器与之对应； 3. 该组寄存器地址中每个偏移地址的含义与“业务级_PF”相应偏移地址含义相同，只是对应VF0； 4. “命令寄存器的接口地址”这一段偏移地址在这里是保留地址。
		PF访问该VF的业务配置	
		PF访问该VF的状态、业务维测	
		该业务资源做业务时的命令 PF看不到这段地址	
	PF_VF1~3		
	PF_VF4	PF访问该VF的中断	该组寄存器地址中每个偏移地址的含义与“PF_VF0”相应偏移地址含义相同，只是对应VF4。
		PF访问该VF的业务配置	
		PF访问该VF的状态、业务维测	
		该业务资源做业务时的命令 PF看不到这段地址	
VF0	业务级_VF0	模块级寄存器（该VF只读）	该组寄存器是“模块级”这组寄存器的重映射。



VF/ PF 制式	模块寄存器地址接口	说明	备注
		该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射； 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同，不同于“PF_VF0”，不是保留地址。
		该VF处理业务时的配置	
		该VF处理业务时的维护统计	
		该VF处理业务时的命令	
VF1~3			
VF4	业务级_VF4	模块级寄存器（该VF只读）	该组寄存器是“模块级”这组寄存器的重映射。
		该VF处理业务时的中断	1. 该组寄存器地址是“PF_VF0”这组寄存器的重映射。 2. 这里“命令”寄存器地址是VF实现业务命令的寄存器地址，偏移地址含义与“业务级_PF”的“PF处理PF业务时的命令”相同，不同于“PF_VF0”，不是保留地址。

- PF访问寄存器
软件以PF制式发起的寄存器访问，POE依照“寄存器资源的虚拟化”相关描述完成对应寄存器的访问操作。
- VF访问寄存器
软件以VF0~4制式访问POE中VF0~4制式的寄存器。软件以VF0~4制式发起的寄存器访问，POE依照“寄存器资源的虚拟化”相关描述完成对应寄存器的访问操作，软件可通过VF0~4制式只读访问到模块级寄存器。
SMMU的页表隔离保证了软件的寄存器访问操作到达POE时，寄存器地址域段的访问必须是对应制式发起的，POE不在对寄存器的访问做多模检查。
对于PF/VF0~4制式发起的非对应制式地址的访问操作，POE只反馈总线操作（读操作反馈全0数据），保证不挂死总线，不进行寄存器的读写操作。
- 包任务/消息添加操作的多模隔离
POE对所有包任务/消息的添加进行多模隔离，保证各制式的包任务/消息不相互影响。在包任务/消息添加到POE时，POE进行了以下的多模隔离操作：
 - POE给每种制式都提供一组独立的包任务/消息添加地址接口，由SMMU的页表隔离保证各制式访问接口的多模隔离。



- POE对添加包任务/消息的操作地址译码出操作的VFID，包任务/消息添加到对应VFID制式的队列中。若添加包任务/消息操作的目的虚拟Group的虚拟化属性没有配置（即在寄存器POE_GRP_VP_MAP、POE_GRP_VF_MAP中没有将任何一个物理Group映射到该VF的虚拟Group），POE将该包任务/消息的添加作丢弃处理，并上报中断。
- 若包任务/消息添加，POE内部缓存空间满需要溢出到DDR空间时，POE根据需要溢出操作的队列所在的制式，发起对应制式的溢出DDR操作，将包任务/消息溢出到对应制式的DDR空间。
- 包任务/消息获取操作的多模隔离
POE对所有包任务/消息的获取进行多模隔离，保证各制式的包任务/消息不相互影响。处理单元在获取包任务/消息时，POE进行了以下的多模隔离操作：
 - POE给每种制式都提供一组独立的包任务/消息获取地址接口，由SMMU的页表隔离保证各制式访问接口的多模隔离。
 - 处理单元向POE发起GET_POE操作获取任务时，POE对获取包任务/消息的操作地址译码出操作的VFID，调度该制式中的任务反馈给处理单元，其他制式的任务不参与调度。
 - 软件向POE发起GET_GROUP/GET_QUEUE操作获取任务时，目的Group号是虚拟的Group编号，通过查找寄存器POE_GRP_VP_MAP、POE_GRP_VF_MAP得到获取任务的指定P_Group进行任务的获取，若在寄存器POE_GRP_VP_MAP、POE_GRP_VF_MAP查找不到映射关系，POE反馈无效任务，并上报中断。
- 保序命令的多模隔离
POE对CPU发起的保序命令进行多模隔离，保证各制式的保序命令互不影响。CPU在发起保序命令操作时，POE进行了以下的多模隔离操作：
 - POE给每种制式都提供一组独立的保序命令操作地址接口，由SMMU的页表隔离保证各制式访问接口的多模隔离。
 - 各制式只能处理该制式的保序任务，POE在收到保序命令时，从地址译码出VFID，根据该制式对应VCPU在寄存器POE_LCPU_BIND_INFO存储保序信息进行保序命令处理。

3.5.22 模式单独复位

- POE多模复位
 - POE给每种制式提供独立的模式复位配置寄存器POE_VF_RST_CFG，配置对应需要复位的模式复位寄存器启动VF复位，VF是否复位完成，需要查看状态寄存器POE_VF_RST_DONE。在启动多模复位前，必须配置队列入队使能寄存器POE_Q_IN_EN，禁止当前复位队列接收新的任务，同时软件保证没有新的任务添加到复位队列中；配置Group出队使能寄存器POE_GRP_OUT_EN，禁止所有处理单元从将复位队列读取任务。队列复位完成后，再次配置恢复队列入队使能和Group出队使能，才允许对队列进行读写操作。
 - 在启动多模复位后，复位模式的Group队列禁止入队操作，加速器任务添加反压为真满，CPU任务添加上报中断给对应VCPU；复位模式的Group任务禁止出队，读取任务反馈7F。
 - 复位模式的Group缓存队列任务丢弃，包括片内缓存的任务、DDR中缓存的任务，但不回收任务Buffer。
 - POE硬件逻辑自动丢弃复位制式的解绑定任务并发送SWITCH_NULL命令给POE内部保序模块，清除保序队列中复位Group的任务信息，不回收任务Buffer。

- POE硬件逻辑自动清除复位制式绑定信息，解除LCPU绑定关系，并发送 SWITCH_NULL命令给POE内部保序模块，清除保序队列中绑定任务信息，不回收任务Buffer。

须知

WRR权重配置为0时，调度器调度不出包，在VF复位时，如果权重配置为0，任务调度不出来，复位完成不了。

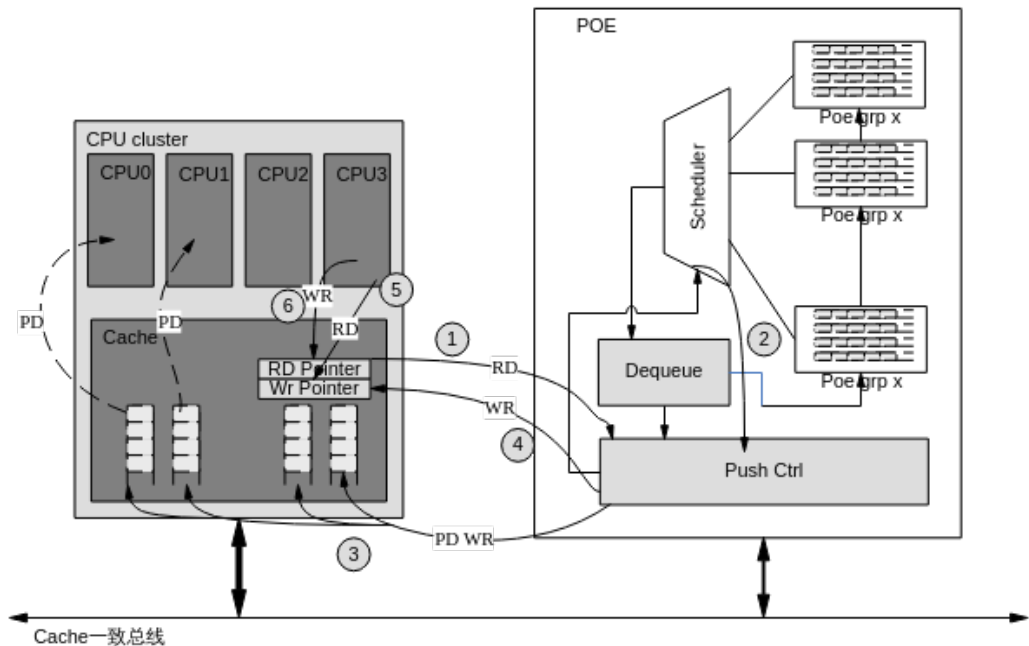
在POE多模复位时，如果是复位SEC的模式任务，需要在mask中将复位的Group去掉；不然SEC在读取此模式的任务时，一直获取不到任务，会引起SEC不停上报中断；其他加速器也需要进行类似上述配置。

3.5.23 推送包到 Cacheable 的存储空间

POE支持推送核间消息或任务到Cacheable的存储空间，支持针对Cacheable的存储空间队列轮询和反压机制（仅限GET_POE操作），用于优化提升CPU获取包任务或消息的性能。

POE通过总线对Cacheable的主存操作与处理器交互。

图 3-17 推送 Cacheable 存储空间示意图



说明1：每个LCPU的队列是私有的；

说明2：如图3-17所示，CPU cluster仅仅是示意，Cache可以为CPU Cluster共享，也可以是CPU独占。

POE通过总线对Cacheable的主存操作与处理器交互。CPU和POE共同在内存中维护16个环形队列，每个队列8深。作为标准的生产者-消费者模型，对于由单个CPU或单Cluster处理的任务，多数情况下，CPU访问可在L2 cache命中，对于跨Cluster处理的



任务，CPU访问可能在L2 cache、其他cluster的L2 cache命中，因此可以提升CPU获取包任务或消息的性能。

LCPU私有Cacheable队列的推送基地址、读写指针地址分别由寄存器POE_PUSH_LCPU_BADDR*、POE_PUSH_LCPU_WR_PTR_ADDR*、POE_PUSH_LCPU_RD_PTR_ADDR*配置。

- 步骤1** POE定时轮询POE_PUSH_LCPU_CACHEQ_EN，为1则执行**步骤2**。
- 步骤2** POE定时读取POE_PUSH_LCPU_RD_PTR，获取RD_Pointer（初始态由CPU在初始化期间完成），并判断Cache私有队列的空满状态。
- 步骤3** 当队列为非满时，POE将为该LCPU以GET_POE方式调度一个任务。
- 步骤4** 该核间消息或者包任务推送给指定的Cache私有存储队列。
- 步骤5** WR_Pointer++，将修改过的WR_Pointer写入POE_PUSH_LCPU_WR_PTR_ADDR*寄存器配置值对应的地址，POE保证**步骤4/步骤5**的顺序。
- 步骤6** LCPU通过RD_Pointer和WR_Pointer判断队列是否非空，读取RD_Pointer指向的内存（Cacheable空间）获取包任务，RD_Pointer++，并回写POE_PUSH_LCPU_RD_PTR_ADDR*寄存器配置值对应的地址，软件要保证读任务和回写读指针的先后顺序，需要DMB。

须知

POE推送Cacheable队列时，固定不推送SourcePort为7F的任务，因此在开启推送Cacheable存储队列时，若物理CPU进入WFE。需要软件通知POE物理CPU已进入睡眠，POE在有包任务时会唤醒物理CPU。

同一个LCPU只支持推送自研核私有队列MessageQueue和推送Cacheable队列中的一种模式，通过配置寄存器POE_PUSH_Q_TYPE选择指定模式。

PUSH的Cacheable队列在业务初始化时由软件完成RD_Pointer和WR_Pointer的初始化。

----结束

3.5.24 从 Cacheable 的存储空间获取包

POE支持轮询Cacheable的主存空间，任务有效时，POE从Cache获取CPU要添加的任务，支持针对Cacheable的存储空间的队列的轮询和反压机制，用于优化提升CPU添加包任务或消息的性能。

POE通过总线对Cacheable的主存操作与处理器交互。CPU和POE共同在内存中维护1-16个环形队列，每个队列8深。作为标准的生产者-消费者模型。

LCPU私有Cacheable队列的Pull基地址、读写指针地址分别由寄存器POE_PULL_LCPU_BADDR*、POE_PULL_LCPU_WR_PTR_ADDR*、POE_PULL_LCPU_RD_PTR_ADDR*配置。

操作步骤：

- 步骤1** POE定时轮询POE_PULL_LCPU_CACHEQ_EN，为1则执行**步骤2**。
- 步骤2** POE定时读取POE_PULL_LCPU_WR_PTR，获取WR_Pointer，并判断Cache私有队列的空满状态。



- 步骤3** 当队列为非空时，POE将从该队列中获取一个任务。
- 步骤4** POE将该任务添加到POE对应的任务队列中。
- 步骤5** RD_Pointer++，将修改过的RD_Pointer写入POE_PULL_LCPU_RD_PTR_ADDR*寄存器配置值对应的地址，POE保证**步骤4/步骤5**的顺序。
- 步骤6** LCPU通过RD_Pointer和WR_Pointer判断队列是否非满，向WR_Pointer指向的内存（Cacheable空间）添加任务，WR_Pointer++，并回写POE_PULL_LCPU_WR_PTR_ADDR*寄存器配置值对应的地址，软件要保证添加包任务和回写WR_Pointer的先后顺序，需要DMB。

须知

PULL的Cacheable队列在业务初始化时由软件完成RD_Pointer和WR_Pointer的初始化。

----结束

3.5.25 CPU 正常业务中断

3.5.25.1 物理队列收包中断

POE为每个队列提供一个水线收包中断，中断使能通过POE_Q_INDRAW_PKT_EN按GROUP配置，水线门限通过寄存器POE_Q_DRAW_PKT_TH_TP、POE_Q_DRAW_PKT_TH_IDX配置，各制式都有一套队列收包水线门限配置寄存器，各制式配置当前制式的虚拟队列的水线，POE通过虚拟到物理的映射关系完成物理队列收包水线门限的配置。寄存器POE_Q_DRAW_PKT_TH_TP是队列收包水线的参考值，POE共提供16组参考值，寄存器POE_Q_DRAW_PKT_TH_IDX是队列收包水线的索引值，各个队列通过索引值到寄存器POE_Q_DRAW_PKT_TH_TP中索引到对应的收包水线参考值。

当物理队列的任务个数从小于等于收包水线门限值增加到大于收包水线门限值时，上报一个物理队列收包脉冲中断；当队列任务数量大于0时，提供一个队列超时计数，通过寄存器POE_Q_INDRAW_PKT_TO_TH配置超时门限值，队列超时计数到达队列超时门限时，上报一个物理队列收包脉冲中断。

物理队列的收包中断是按物理Group粒度进行上报，当一个队列的出现收包中断时，根据当前队列所在的物理Group号进行任务上报，POE一个提供16根脉冲中断进行物理队列收包中断上报，每个中断线代表一个物理Group中的队列是否有收包中断。POE仅为逻辑CPU能够获取的物理Group提供物理队列收包中断。POE为每个制式中使用的虚拟队列提供任务数量超过收包水线的状态以及超时状态，通过状态寄存器Q_GET_TH_ST和Q_GET_TH_TO_ST可查询。

3.5.25.2 Group 粒度超时未取包中断

POE为每个Group提供一个未取包超时中断，超时门限通过寄存器POE_GROUP_TMOOUT_CFG配置，当Group中有任务且没有处理单元来该Group获取任务时，开始取包超时计数，当取包超时计数值到达超时中断门限时，上报中断给对应逻辑CPU，中断状态各制式按虚拟CPU记录。软件可查询寄存器VCPU_INT_STATUS.inq_grp_tout_int_status。对应制式的虚拟Group编号可在VCPU_INTS_INFO查询。



3.5.25.3 Group 粒度水线中断

POE为每个Group提供一个水线中断，Group水线门限通过寄存器POE_GRP_INT_TH配置，当Group中任务数量大于等于Group水线中断门限值时，上报中断给对应逻辑CPU，中断状态各制式按虚拟CPU记录。软件可查询寄存器VCPU_INT_STATUS.grp_th_int_status。

3.5.25.4 物理队列高低水线中断

- POE为每个队列提供一个队列高水线中断，高水线门限通过寄存器POE_Q_INT_TH_H_TP、POE_Q_INT_TH_H_IDX配置，各制式都有一套队列高水线门限配置寄存器，各制式配置当前制式的虚拟队列的水线，POE通过虚拟到物理的映射关系完成物理队列高水线门限的配置。寄存器POE_Q_INT_TH_H_TP是队列高水线的参考值，POE中每个VF0~4各提供16组参考值可配置，寄存器POE_Q_INT_TH_H_IDX是队列高水线的索引值，各个队列通过索引值到寄存器POE_Q_INT_TH_H_TP中索引到对应的高水线参考值。
- POE为每个队列提供一个队列低水线中断，低水线门限通过寄存器POE_Q_INT_TH_L_TP、POE_Q_INT_TH_L_IDX配置，各制式都有一套队列低水线门限配置寄存器，各制式配置当前制式的虚拟队列的水线，POE通过虚拟到物理的映射关系完成物理队列低水线门限的配置。寄存器POE_Q_INT_TH_L_TP是队列低水线的参考值，POE中每个VF0~4各提供16组参考值可配置，寄存器POE_Q_INT_TH_L_IDX是队列低水线的索引值，各个队列通过索引值到寄存器POE_Q_INT_TH_L_TP中索引到对应的低水线参考值。

当队列任务数量大于队列高水线值时，该队列是CPU处理的队列时上报中断给各制式中对应逻辑CPU，该队列是加速器处理的队列时上报中断给各制式的VF；当队列任务数量大于队列高水线值后再低于队列低水线时，该队列是CPU处理的队列时上报中断给各制式中对应逻辑CPU，该队列是加速器处理的队列时上报中断给各制式的VF。POE为每种制式中使用的虚拟队列提供队列水线状态寄存器Q_INT_TH_ST，队列中任务个数大于队列高水线门限时，Q_INT_TH_ST[q_int_th_st_h]状态有效为1，当队列中任务个数低于队列低水线门限时，Q_INT_TH_ST[q_int_th_st_l]状态有效为1。

高低水线中断，软件可分别查询寄存器域值VCPU_INT_STATUS.cpu_q_th_h_int_status和VCPU_INT_STATUS.cpu_q_th_l_int_status。具体每个队列的状态可以查询寄存器Q_INT_TH_ST。

须知

- 当队列的高低水线门限配置为相等值时“X”时，当任务个数大于“X”时，上报高水线中断，且在任务个数大于“X”后由于取包操作导致任务个数小于“X”时，上报低水线中断；
- 当高低水线中断用于流控的配置约束，需要保证队列的高水线门限大于低水线门限；
- 当高低水线中断用于收包的配置约束：低水线配置为1，高水线配置为需要的值；若需要来包就上报则需要将低水线配置为1，高水线配置为0。

3.5.26 原子特性

对于64bit Timer，需要支持原子特性，确保Master读取计时信息的正确性。

每个64bit Timer内部有1组record寄存器，当收到某个Master的读操作时，根据地址返回32bits数据，另外32bits数据寄存在record寄存器，等待下次同一Master的读取。



多个64bit Timer之间可以通过同一个使能寄存器开启计数。当多个64bits Timer的初始值设定一致后，配置该使能寄存器，多个64bits Timer可以同时启动。

当有多个CPU或者多个线程访问Timer时，需要软件加锁，保证一个CPU完整访问后才允许下一个CPU访问。

须知

Timer中对64bit寄存器（Timer64_VALUE_L和Timer64_VALUE_H）支持单线程原子操作。

原子操作机制：

- 一次全新的读访问，如果操作低32位寄存器后，则同时将高32位寄存器值保持在模块内部，在下次访问高32位数据时，返回高位数据；如果继续访问低位数据，则返回更新后的低位数据，重新锁存高32位数据。同理，如果先访问高32位数据，则同时会将低32位数据保存在模块内部，在下次访问低32位数据时返回该值。在一次连续的低32位和高32位被访问之后，则本次原子访问操作结束。
- 一次全新的写访问，只有在低32位和高32位被连续写访问后，才会将64位数据更新到寄存器中。

3.6 软件指南

3.6.1 大小端转换

- POE中的大小端支持如下：
 - 普通寄存器仅支持小端模式的访问。
 - CPU进行命令寄存器访问支持大小端可配。
 - POE推送功能，推送给加速器的任务仅支持小端模式，推送给CPU的任务支持大小端模式可配置。
 - POE支持按制式配置大小端模式。
 - POE支持按制式配置32bit大小端转换还是64bit大小端转换。
- POE中的大小端处理：
 - POE内部存储固定小端存储。
 - CPU访问命令寄存器的大小端模式通过各自制式寄存器POE_BIGENDIAN_CFG配置。
 - 如果CPU访问命令寄存器是大端模式，POE接口将CPU添加的数据转换为小端形式存储，将CPU读取数据转换为大端形式给CPU。

如果CPU访问命令寄存器是小端模式，POE接口对数据不再转换。



表 3-5 32bit 大小端转变表（A4/B4/C4/D4 为 MSB）

小端																															
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
A4								A3								A2								A1							
B4								B3								B2								B1							
C4								C3								C2								C1							
D4								D3								D2								D1							
大端																															
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
A1								A2								A3								A4							
B1								B2								B3								B4							
C1								C2								C3								C4							
D1								D2								D3								D4							

表 3-6 64bit 大小端转变表（A8/B8 为 MSB）

小端																																
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4																									
A4								A3								A2								A1								
A8								A7								A6								A5								
B4								B3								B2								B1								
B8								B7								B6								B5								
大端																																
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0											
A5								A6								A7								A8								
A1								A2								A3								A4								
B5								B6								B7								B8								
B1								B2								B3								B4								



须知

POE大小端模式默认为大端模式访问。

3.6.2 数据结构

本节内容描述POE模块所有的由处理单元发给POE的命令数据结构：

- GET_POE/GET_GROUP/GET_QUEUE命令。
- ADD_PACKET命令。
- SWITCH_NULL/ SWITCH_DESCCHEDULE/ DESCCHEDULE命令。
- SEND_TM/ SEND_POE命令。
- FORWARD_TM命令。

POE给各加速器的任务状态的消息数据结构。

须知

- POE标签页的相关数据结构中，Reserve域软件不可自定义填写，是保留域。User_Data域是软件可自定义填写的域。
- CPU和加速器之间， 加速器和加速器之间通过POE交互的数据流要求任务数据结构中的word7固定保留用于Hardware TLV。

3.6.3 寄存器动态配置

- P_Group与V_Group映射关系禁止动态配置
 - P_Group与V_Group的映射关系配置寄存器POE_GRP_VL_MAP、POE_GRP_VP_MAP，由PF制式配置，在各制式初始化阶段需要静态配置好映射关系。
 - P_Group与V_Group的映射关系需要重新配置时，确保当前重配置的P_Group没有待处理的任务，才能由PF制式重新配置。
- Group保序/非保序动态配置
 - 逻辑CPU任务P_Group保序/非保序动态配置时，需所有逻辑CPU暂停向POE获取任务。配置后，逻辑CPU获取的任务按最新配置处理。

须知

如果P_Group从保序动态配置为非保序后，逻辑CPU可能还会读取到SWTICH_DESCCHEDULE执行后的保序任务，需对任务进行保序流程操作。

- P_Group中的核间消息不受保序、非保序的限制，固定不保序。
- Group负载均衡动态配置
 - 逻辑CPU任务P_Group动态调整时，保证POE中残留的任务有逻辑CPU处理，保证POE中不会出现任务拥塞。



- 内部加速器任务P_Group动态调整时，需要先查询GROUP_CNT，确认当前P_Group中没有待处理的任务，然后才能切换，保证业务处理流程正确。
- 调度算法动态配置
 - 调度算法相关配置后立即生效，处理单元获取任务时按最新调度配置调度，WRR的权重值配置在本轮调度完后生效。
- POE溢出空间禁止动态配置
 - DDR溢出空间按制式独立配置，每种制式在使用溢出空间前，单独初始化该制式的溢出空间，所有制式的溢出空间在初始化时配置，禁止动态配置。
- Group反压水线、VF水线、保序队列水线可动态配置
 - 水线配置后立即生效，POE按最新配置进行处理。
- 其他未描述的寄存器禁止动态配置。

3.6.4 POE 初始化

3.6.4.1 模块级初始化

POE在上电复位后，在任何制式使用之前，都需进行模块级初始化：

- 步骤1** 确保已经按照《Hi309a 通用MPU芯片 用户指南01-CPU小系统》完成了初始化POE模块之前的其他初始化步骤。
- 步骤2** 软件配置POE的memory初始化使能寄存器，POE_INIT_RAM_START，开启内部memory自动初始化操作。
- 步骤3** 软件等待5μs，检查POE内部RAM初始化是否完成，检查状态寄存器POE_INIT_RAM_DONE。如果寄存器值为0，表示仍在初始化，如果寄存器值为1，表明初始化已完成。
- 步骤4** 配置芯片公共寄存器SR_IOV_CTL、SR_IOV_VF_OFFSET。
- 步骤5** 配置寄存器POE_PF_STREAM_ID、POE_VF_STREAM_ID、POE_VF_SSET。
- 步骤6** 配置寄存器POE_CPU_VL_MAP、POE_CPU_VF_MAP，配置逻辑CPU与虚拟CPU的映射关系。
- 步骤7** 配置寄存器POE_GRP_VP_MAP、POE_GRP_VF_MAP，配置物理Group与虚拟Group的映射关系。
- 步骤8** 配置寄存器POE_LCPU_MAP。
- 步骤9** 软件可选择配置POE_VQID_ROUTE_BYPASS_CFG，使能Bypass VQI路由表，直接使用VQI作为虚拟Qid。
- 步骤10** 按需要配置模块中断使能寄存器POE_PF_COM_INT_ENABLE。
- 步骤11** 配置POE_GROUP_AFFINITY_EN选择开启流分类功能的GROUP。
- 步骤12** 按需要配置寄存器POE_CPU_WFE_TMOUT_CFG，配置POE逐个唤醒物理CPU的时间间隔。
- 步骤13** 按业务需要配置POE中超时门限类寄存器：
1. POE_VCPU_ADD_RETRY_TIME_CFG。
 2. POE_GROUP_TMOUT_CFG。



3. POE_ATOMIC_TMOUT_CFG。
4. POE_PUSH_CPU_TMOUT_CFG。
5. POE_PULL_CPU_TMOUT_CFG。
6. POE_PUSH_CACHEQ_WFE_TMOUT_CFG。
7. POE_Q_INDRAW_PKT_TO_TH。
8. POE_VF_RST_DLY_GAP_CFG。

步骤14 按业务需要配置POE中访问DDR及Cache的OTD深度寄存器：

1. POE_OVFL_OTD_NUM。
2. POE_RDBK_OTD_NUM。
3. POE_OVRB_OTD_NUM。
4. POE_PUSH_RD_PTR_OTD_NUM。
5. POE_PUSH_WR_PTR_OTD_NUM。
6. POE_PUSH_PACKET_OTD_NUM。
7. POE_PULL_RD_PTR_OTD_NUM。
8. POE_PULL_WR_PTR_OTD_NUM。
9. POE_PULL_PACKET_OTD_NUM。

步骤15 配置内部缓存队列共享配置寄存器。

1. POE_VF_MAX_NUM。
2. POE_Q_MAX_TP。
3. POE_Q_MIN_TP。
4. POE_Q_MIN_INDEX。

步骤16 配置加速器任务通知消息功能相关的寄存器：

1. POE_MSG_TMOUT_CFG。
2. POE_MSG_TMOUT_EN。
3. POE_SEND_MSG_OTD_NUM。
4. POE_MSG_ACC_PF_ADDR_H_CFG。
5. POE_MSG_ACC_PF_ADDR_L_CFG。
6. POE_MSG_ACC_VF_OFFSET_EN。
7. POE_MSG_ACC_VF_OFFSET。

步骤17 配置SEND_TM / SEND_POE相关寄存器：“*”代表各个加速器名称。

1. POE_SEND_*_OTD_NUM。
2. POE_SEND_*_VF_ADDR_H_CFG。
3. POE_SEND_*_VF_ADDR_L_CFG。
4. POE_SEND_*_VF_OFFSET。
5. POE_SEND_*_STRIDE_SIGN。

步骤18 配置FORWARD_TM相关寄存器：“*”代表各个加速器名称。

1. POE_FORWARD_*_OTD_NUM。
2. POE_FORWARD_*_VF_ADDR_H_CFG。



3. POE_FORWARD_*_VF_ADDR_L_CFG。
4. POE_FORWARD_*_VF_OFFSET。
5. POE_FORWARD_*_STRIDE_SIGN。

步骤19 配置释放BMU操作相关的寄存器：

1. POE_SEND_BMU_OTD_NUM。
2. POE_SEND_BMU_VF_SPADDR_H_CFG。
3. POE_SEND_BMU_VF_SPADDR_L_CFG。
4. POE_SEND_BMU_VF_OFFSET。
5. POE_SEND_BMU_STRIDE_SIGN。

步骤20 配置推送LCPU的物理地址配置寄存器：

1. POE_PUSH_LCPU_BADDR_H。
2. POE_PUSH_LCPU_BADDR_L。
3. POE_PULL_LCPU_BADDR_H。
4. POE_PULL_LCPU_BADDR_L。
5. POE_PUSH_LCPU_RD_PTR_ADDR_H。
6. POE_PUSH_LCPU_RD_PTR_ADDR_L。
7. POE_PUSH_LCPU_WR_PTR_ADDR_H。
8. POE_PUSH_LCPU_WR_PTR_ADDR_L。
9. POE_PULL_LCPU_RD_PTR_ADDR_H。
10. POE_PULL_LCPU_RD_PTR_ADDR_L。
11. POE_PULL_LCPU_WR_PTR_ADDR_H。
12. POE_PULL_LCPU_WR_PTR_ADDR_L。

步骤21 配置加速器到加速器的到达保序相关配置寄存器：

1. POE_SEND_CREDIT_OTD_NUM。
2. POE_SEND_CREDIT_ADDR_H。
3. POE_SEND_CREDIT_ADDR_L。
4. POE_HAC_ADD_CHN_INFO[hac_add_chn_cdt_th]。

步骤22 根据需要配置反压TM相关配置寄存器：

1. POE_BP_*_GRP_TH_*。
2. POE_BP_*_CFG。

步骤23 配置寄存器中“模块级配置寄存器”中尚未配置的业务需要的寄存器。

----结束



须知

- 软件在配置唤醒间隔寄存器POE_CPU_WFE_TMOUT_CFG时，如果时间间隔配置过大，大流量时会影响芯片处理性能；如果时间间隔配置过小时，会影响芯片的功耗。
- POE_SLV_DAW1_ADDR默认值支持TOTEM_B，如果需要支持TOTEM_A，需要在修改其配置值。
- WRR权重配置为0时，调度器调度不出包，在VF复位时，如果权重配置为0，任务调度不出来，复位完成不了。
- 建议POE_HAC_ADD_CHN_INFO[hac_add_chn_cdt_th]寄存器配置为加速器ADD_POE_outstanding深度减1。

3.6.4.2 业务级初始化

POE在上电复位后或软复位之后，在完成模块级初始化之后，按需要对各个制式的业务相关寄存器进行配置，步骤如下（如果启动多个VF业务，则重复执行此流程）：

- 步骤1** 配置寄存器POE_BIGENDIAN_CFG，确定本制式下所有CPU对POE的加包和获取包操作的大小端模式，配置寄存器POE_PUSH_PULL_ENDIAN，设置制式下POE推送和获取LCPU私有队列的大小端模式。
- 步骤2** 按业务需要配置寄存器POE_GRP_VCPU_MASK、POE_GRP_ACC_MASK，配置完成后查询WR_MASK_FAIL寄存器确认是否有配置MASK失败的状态，若出现配置失败，请参考“[负载均衡配置](#)”中对配置失败后的处理步骤。
- 步骤3** 按业务需要配置调度策略相关寄存器。
1. POE_Q_WRR_CFG。
 2. POE_Q_SCH_QOS_CFG。
- 步骤4** 按业务需要配置GROUP的相关配置寄存器。
1. POE_GRP_OUT_EN。
 2. POE_GRP_ORDER_TG。
 3. POE_GRP_ORDER_FLOW_EN。
 4. POE_ATOMIC_MASK_EN。
 5. POE_SCHE_MODE_CFG。
- 步骤5** 按业务需要配置内部缓存队列相关配置寄存器。
1. POE_Q_IN_EN、POE_IN_Q_LEN。
 2. POE_IN_Q_MIN_RSV_EN。
 3. POE_IN_Q_MAX_INDEX。
- 步骤6** 按业务需要配置转发队列相关配置寄存器。
1. POE_VF_FWD_OVFL_EN。
 2. POE_FWD_Q_MIN_RSV_EN。
 3. POE_FWD_Q_MAX_INDEX。
- 步骤7** 按业务需要配置使用POE的溢出。
1. 空间大小（寄存器POE_OVER_LEN）。



2. 配置对应的虚拟Group的溢出使能寄存器POE_GRP_OVFL_EN。
3. 配置POE_OVER_BASE_H_ADDR、POE_OVER_BASE_L_ADDR。

步骤8 配置业务中断POE_VCPU_INT_ENABLE和POE_INT_ENABLE两个中断使能寄存器。

步骤9 按业务需要配置POE中水线类配置寄存器。

1. POE_Q_INT_TH_H_IDX。
2. POE_Q_INT_TH_L_IDX。
3. POE_Q_INDRAW_PKT_TH_IDX。
4. POE_Q_INT_TH_H_TP。
5. POE_Q_INT_TH_L_TP。
6. POE_Q_INDRAW_PKT_TH_TP。
7. POE_GRP_INT_TH。
8. POE_GRP_BP_TH。
9. POE_VF_INT_TH。
10. POE_INFLIGHT_TH。
11. POE_Q_INDRAW_PKT_EN。

步骤10 按业务需要配置PUSH LCPU私有队列相关寄存器，PUSH的Cacheable队列的读写指针由软件初始化。

POE_PUSH_LCPU_MAX_NUM。

步骤11 按业务需要配置PULL LCPU私有队列，PULL的Cacheable队列的读写指针由软件初始化。

步骤12 按业务需要CPU WFE唤醒模式寄存器。

POE_CPU_WFE_MODE_CFG。

步骤13 按业务需要VQI映射配置寄存器POE_VQID_MAP；按业务需要配置POE发送任务通知命令给各加速器的使能配置寄存器POE_MSG_ACC_EN。

步骤14 配置nManger中“业务配置寄存器”中尚未配置的业务需要的寄存器。

步骤15 配置VF业务初始化寄存器POE_VF_INIT_EN。

步骤16 软件等待1ms，检查VF初始化是否完成，检查状态寄存器POE_VF_INIT_DONE。如果寄存器值为0，表示仍在初始化，如果寄存器值为1，表明初始化已完成。

步骤17 软件根据需要开启LCPU各私有队列使能，开启POE与LCPU的队列交互。

1. POE_PULL_LCPU_CACHEQ_EN。
2. POE_PUSH_LCPU_CACHEQ_EN。

----结束

须知

VF的溢出资源在软件配置了POE_VF_INIT_EN后自动完成初始化，不需要额外配置。



3.6.4.3 独立软复位

在系统工作允许的情况下，可以任意时间对POE工作逻辑进行软复位，但软件需要保证不会对系统造成影响。通过配置系统控制器，实现对POE的Master时钟域工作逻辑的单独软复位。

安全复位流程：

- 步骤1** 将POE模块寄存器POE_DEBUG_MODE_EN配置为0x5A5A。
- 步骤2** 将POE模块Master口使能配置寄存器（POE_MST_FUN_EN）配置为0（不使能Master接口）。
- 步骤3** 配置所有LCPU的POE_PUSH_LCPU_CACHEQ_EN、POE_PULL_LCPU_CACHEQ_EN，关闭所有LCPU PUSH和PULL使能；需要配置POE发送任务通知命令给各加速器的使能配置寄存器POE_MSG_ACC_EN的为0，关闭任务通知。
- 步骤4** 软件确保CPU不再对POE进行读写访问（软件解决方案：配置POE软复位寄存器和解复位寄存器前后都添加DSB指令），确保与POE模块SLAVE接口交互的各加速器的Master接口为空闲状态。
- 步骤5** 确认CPU和各加速器对POE的所有读写访问已经完成。
- 步骤6** 确认POE工作状态寄存器POE_MST_FUN_IDLE中Master口工作状态为空闲状态。
- 步骤7** 配置系统控制寄存器SC_POE_CLK_DIS['icg_dis_poe_slv]和['icg_dis_poe_mstr]关闭时钟。
- 步骤8** 配置系统控制寄存器SC_POE_RESET_REQ对POE进行软复位。
- 步骤9** 配置系统控制寄存器SC_POE_CLK_EN['icg_en_poe_slv]和['icg_en_poe_mstr]开启模块时钟，触发模块复位，延时1ms。
- 步骤10** 配置系统控制寄存器SC_POE_CLK_DIS['icg_dis_poe_slv]和['icg_dis_poe_mstr]关闭时钟。
- 步骤11** 配置系统控制寄存器SC_POE_RESET_DREQ解除POE复位。
- 步骤12** 查询SC_POE_RESET_ST，若查询值为0，则表示模块处于解复位状态。
- 步骤13** 配置系统控制寄存器SC_POE_CLK_EN['icg_en_poe_slv]和['icg_en_poe_mstr]开启模块时钟。
- 步骤14** 查询SC_POE_CLK_ST，若查询值为0x3，则表示模块时钟打开。
- 步骤15** 将POE模块Master口使能配置寄存器（POE_MST_FUN_EN）配置为1（使能Master接口）。
- 步骤16** 将POE模块寄存器POE_DEBUG_MODE_EN配置为0（关闭debug模式，防止误操作）。

----结束



须知

1. 如果复位后又重新被用于业务，一定要重新进行模块级初始化和业务级初始化。
2. 若软件需要复位配置寄存器，需要软复位整个POE。
3. 配置POE模块Master口使能配置寄存器（POE_MST_FUN_EN）时，必须先将POE模块寄存器POE_DEBUG_MODE_EN配置为16h5a5a。
4. 对各个添加POE的加速器来说，如果某个加速器被复位了，再次启动使用之前，需要把该加速器对应的POE中POE_HAC_ADD_CHN_CFG寄存器对应位配置为1（需要先将寄存器POE_DEBUG_MODE_EN配置为16h5a5a）。
5. 对各个POE推送的加速器来说，如果某个加速器被复位了，再次启动使用之前，需要把该加速器对应的POE的POE_GET_OTD_SN_INIT_EN[fwd_acc_otd_sn_init_en]和POE_GET_OTD_SN_INIT_EN[send_acc_otd_sn_init_en]寄存器值写1，初始化添加任务的SN号（需要先将寄存器POE_DEBUG_MODE_EN配置为16h5a5a）。
6. 对各个从POE获取任务的加速器来说，如果某个加速器被复位了，再次启动使用之前，需要把该加速器对应的POE的POE_GET_OTD_SN_INIT_EN[get_acc_otd_sn_init_en]寄存器值写1，初始化获取任务的SN号（需要先将寄存器POE_DEBUG_MODE_EN配置为16h5a5a）。
7. 对各个POE推送的加速器来说，如果POE被复位了，再次启动使用之前，需要重新初始化相应的加速器的SN号为0。
8. 对各个从POE获取任务的加速器来说，如果POE被复位了，再次启动使用之前，需要重新初始化相应的加速器的SN号为0。

3.6.4.4 模式独立复位

在系统工作允许的情况下，软件可以通过配置POE中多模复位对应的寄存器，实现对POE的多模复位。

多模复位的流程：

- 步骤1** 在启动多模复位之前，软件确保需要多模复位的业务全部停止，保证需要复位模式的CPU/加速器都处于空闲状态，即正在处理的任务处理完成，CPU/加速器不再从POE获取任务，CPU/加速器不再向POE添加任务。
- 步骤2** 配置待复位VF的加速器MASK寄存器POE_GRP_ACC_MASK，将要复位的Group从MASK寄存器上移除，避免加速器不停上报获取不到任务的中断。
- 步骤3** 配置待复位VF的POE_PUSH_LCPU_CACHEQ_EN、POE_PULL_LCPU_CACHEQ_EN，关闭待复位VF下的LCPU PUSH和PULL使能。
- 步骤4** 配置POE_Q_IN_EN寄存器，禁止需要多模复位的队列接收新任务，同时软件保证没有新的任务添加到需要待复位的队列。
- 步骤5** 配置POE_GRP_OUT_EN寄存器，禁止所有处理单元从需要待复位的队列读取任务。
- 步骤6** 配置对应制式的POE_VF_RST_CFG寄存器为1，启动模式复位。
- 步骤7** 软件等待1ms后，查询复位完成状态，读取POE_VF_RST_DONE寄存器，查看多模复位是否已经完成。
- 步骤8** 软件需要重新进行业务级初始化，才能对完成复位后的制式重新使用。

----结束



须知

- 单制式在POE中的任务总数最多约5M个256bit的包任务，在模式复位时，每25个POE时钟丢弃一个包任务，约1ms。
- VF单模复位都不会复位业务级配置寄存器，如果该VF单模复位后又重新被用于业务，一定要重新进行业务级初始化。
- 模式复位不会复位中断状态、中断统计（如果有的话）寄存器，单模式复位后需要软件读清中断状态，读清中断统计计数器。

3.7 性能

3.7.1 吞吐量（Throughput）

ADD/GET POE性能数据（吞吐量，按典型600MHz计算，且基于如下评估模型计算）。

- POE提供的单队列最大加包性能是：80Mpps（128bit任务描述符），50Mpps（256bit任务描述符）。
- POE提供的多队列最大加包性能是：80Mpps（128bit任务描述符），50Mpps（256bit任务描述符）。
- POE提供的单队列最大取包性能是：60Mpps（256bit非保序任务），15Mpps（256bit保序任务）。
- POE提供的多队列最大取包性能是：60Mpps（256bit非保序任务），15Mpps（256bit保序任务）。
- POE提供的SEND_TM最大推包性能是：15Mpps（按照一次推包访问时延200ns计算）。
- POE提供的FORWARD_TM最大推包性能是：15Mpps（按照一次推包访问时延200ns计算）。
- POE提供的SEND_POE最大推包性能是：15Mpps。

3.7.2 延迟（Latency）

ADD/GET POE性能数据（时延）：

- 向POE添加一个任务/保序命令的无阻塞延时是15个POE工作时钟。
- 从POE中指定队列读取一个任务的无阻塞延时是23个POE工作时钟。
- 从POE读取一个非保序任务的无阻塞延时是23个POE工作时钟。
- 从POE读取一个保序任务的无阻塞延时是27个POE工作时钟。

3.8 DFX 应用指导

3.8.1 可靠性



3.8.1.1 CPU 加包防止错误传播

POE接收到CPU的添加128bit包任务，当CPU发出32bit写时，一个完整的包需要4次写操作方可收集齐，可能出现某个32bit加包地址被误踩踏这样的错误。

为了避免出现上述错误的传播，一次踩踏只影响当前包数据，不会影响后续包的正确接收，所以POE按照以下方法进行收包。

128bit包的组成是{word3, word2, word1, word0}，这个方法的前提是CPU发包时到达POE的word顺序一定是这样的：如果是32bit写操作，POE的slave接口收到的先后顺序是word0, word1, word2, word3。如果是64bit写操作，POE的slave接口收到的先后顺序是{word1, word0}, {word3, word2}。即CPU按顺序发出写指令，这些指令一定要在POE的slave侧到达保序，才能保证该方法的有效性。

POE接收到CPU的加包数据时，按照写命令地址来进行收包：

CPU发送128bit保序命令或消息时：

- 32bit的写，POE接收到一个包任务的WORD3地址对应的数据，即认为收齐了128bit数据，并将这128bit数据写入任务缓存。
- 64bit的写，POE接收到一个包任务的WORD2地址对应的数据，即认为收齐了128bit数据，并将这128bit数据写入任务缓存。
- 128bit的写，POE接收到一个包任务的WORD0地址对应的数据，即认为收齐了128bit数据，并将这128bit数据写入任务缓存。

CPU发送256bit消息时：

- 32bit的写，POE接收到一个包任务的WORD7地址对应的数据，即认为收齐了256bit数据，并将这256bit数据写入任务缓存。
- 64bit的写，POE接收到一个包任务的WORD6地址对应的数据，即认为收齐了256bit数据，并将这256bit数据写入任务缓存。
- 128bit的写，POE接收到一个包任务的WORD4地址对应的数据，即认为收齐了256bit数据，并将这256bit数据写入任务缓存。

3.8.1.2 保序队列队头超时异常处理

保序队列中，如果队头任务长时间没有处理单元来处理，上报中断给VF。

步骤1 VF通过查询对应中断状态信息，获取超时的队头对应的Entry ID。

步骤2 VF通过轮询并比较16个VCPU绑定信息中的Entry ID，获取超时的队头保序任务所绑定LCPU。

步骤3 若超时任务已绑定VCPU，建议VF通过冒充VCPU发送DESCCHEDULE命令将队头任务放到解绑定缓存中，供其他VCPU获取处理。

步骤4 若超时任务未绑定VCPU，则该超时保序任务在解绑定队列中或者该任务已不在保序队列中，建议软件不做额外处理。

----结束

3.8.1.3 CPU 加包异常

CPU添加任务到POE时，会出现添加失败的情况出现，添加任务到POE，失败的可能场景有如下：



- 逻辑CPU添加任务的虚拟Group号错误：
 - 逻辑CPU添加的任务的目的V_Group在寄存器POE_CPU_VL_MAP、POE_GRP_VP_MAP中未有配置，即当前配置的制式并没有配置使用该V_Group号，POE不接收该任务的添加。
- 逻辑CPU添加任务的VCPU号错误：
 - VCPU无法映射到LCPU的错误，POE不接收任务的添加和获取。
- 逻辑CPU添加任务，V_Group号未配置使用，或逻辑CPU添加任务的队列号错误，POE不接收该任务的添加，任务丢弃，上报中断给对应制式的逻辑CPU，添加失败的任务写入失败缓存空间，按制式记录失败次数POE_VCPU_ADD_GQ_ERR_CNT寄存器。

失败任务缓存空间：

- 5种制式，每个制式有16深、128bit宽的缓存来存储添加失败任务，当缓存满时，后续对添加失败任务进行丢弃处理。
- 128bit宽的失败缓存数据中bit[117:0]缓存包任务添加时数据结构的bit[117:0]，bit[127:118]用于记录出错的错误类型及信息。
- 软件可通过寄存器接口VCPU_FAIL_BUF0_GET~VCPU_FAIL_BUF3_GET读取对应制式的失败缓存任务，读取FAIL_BUF的操作，需要以Single 128bit的读取操作进行访问，否则会出现读不到完整的数据的情况。
- 失败任务缓存每个数据单元的数据结构如表3-7所示。

表 3-7 POE 失败任务数据结构

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
PKT_Data																														
PKT_Data																														
PKT_Data																														
Err_Ty pe		Vcpu_id						PKT_Data																						
说 明																														
PKT_Data[117:0]		出现真满、队列禁止、Group/Qid非法时，记录所添加任务的低118bit号。																												
vcpu_id[6:0]		记录添加失败的VCPU编号0~63。																												



3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
Err_Type[2:0]						错误类型： 000：真满； 001：路由失败； 010：对禁止入队的队列进行添加； 011：添加任务时Group/Qid非法； 100：添加任务ECC 2bit校验错误； 101：CPU添加任务假满重试超时； 110：CPU添加任务包长异常或者TLV异常； 其他值都是保留值。																									

3.8.1.4 加速器加包异常

请参见“[加速器加包（ADD_PACKET）](#)”中加速器加包失败描述。

3.8.1.5 物理队列的满中断

实际上这个就是物理队列的真满中断，即该队列在片内空间和DDR溢出空间全满；
CPU所属队列按照VCPU为粒度上报中断POE_VCPU_INT_STATUS.
cpu_add_real_full_int_status，加速器所属队列按照PF/VF为粒度上报中断
POE_INT_STATUS.hac_add_real_fullint_status。

3.8.1.6 模式粒度的水线中断

POE为每种制式提供一个水线中，VF水线门限通过寄存器POE_VF_INT_TH配置，当制式中任务数量大于等于制式水线中断门限值时，上报中断给对应制式，按制式记录中断状态POE_INT_STATUS.vf_th_int_status。

3.8.1.7 接口命令错误

逻辑CPU执行GET_POE、GET_GROUP命令，从POE中获得包处理，并且包需要保序，包信息入保序队列，此时处理单元才能和包任务建立绑定关系，否则无绑定关系。

- 有绑定关系的逻辑CPU，如果再发送GET_POE、GET_GROUP命令给POE，反馈获取任务失败。此时，逻辑CPU可以发送SWITCH_NULL、SWITCH_DESCCHEDULE、DESCCHEDULE、SEND_TM、SEND_POE、ADD_PACKET、命令给POE，POE按命令执行操作。
- 无绑定关系的逻辑CPU，在发送GET_POE、GET_GROUP、GET_QUEUE、ADD_PACKET、命令给POE时，POE按命令执行操作。在发送SWITCH_NULL、SWITCH_DESCCHEDULE、DESCCHEDULE、SEND_TM、SEND_POE命令时，POE忽略这些命令，上报中断，中断状态在寄存器POE_VCPU_INT_STATUS.abn_op_int_status记录。

3.8.1.8 内部数据完整性错误

- POE处理RAM的ECC校验，ECC 1bit校验错误时，POE上报中断到PF，对应中断状态在寄存器POE_PF_COM_INT_STATUS.ecc_1bit_err_int_status，并且记录发生



ECC 1bit对应的RAM编号在寄存器POE_ECC_1BIT_ERR_INFO。ECC 1bit校验错误，POE对发生ECC 1bit错误的数据进行纠错处理，将纠错后的数据继续进行处理，不影响POE的数据流处理。

- POE处理RAM的ECC校验，ECC 2bit校验错误时，POE上报中断到PF，对应中断状态在寄存器POE_PF_COM_INT_STATUS.ecc_2bit_err_int_status，并且记录发生ECC 1bit对应的RAM编号在寄存器POE_ECC_2BIT_ERR_INFO。ECC 2bit校验错误时，会造成POE中数据的丢失，需要软件进行软复位操作。

3.8.2 模块维侧

3.8.2.1 Debug

本模块提供了以下方式用于内部状态查询，以方便系统调试（Debug）。

- 本模块提供给了Debug Bus接口，该接口可配置和可读取RAM寄存器配置信息及统计信息。
- POE内部的所有寄存器，均可以通过Debug Bus接口进行读写访问。

须知

- Debug Bus接口支持访问POE的16*9M地址空间且不支持多模，采用地址的低28bit译码，地址的bit[23:0]为各空间的寄存器偏移地址。
- 当Debug Bus接口的访问操作与AXI总线的访问操作同时发生时，Debug Bus操作不被执行，POE反馈错误响应给Debug Bus接口。

3.8.2.2 POE 内部 RAM 软件可访问

在Debug模式，软件可以通过间接地址访问的方式读取POE的表项内容。软件在寄存器DEBUG_RAM_NUM和DEBUG_RAM_ADDR中写入要访问的RAM编号和访问entry地址，即可通过读取DEBUG_RD_RAM_OP获取对应entry的内容，最后查询DEBUG_RD_RAM_ST可判断间接访问的数据是否已成功准备好并放入寄存器DEBUG_RD_RAM_DATA*。

3.8.2.3 ECC 错误注入

通过配置POE_RAM_ECC_INJECT_REQ*寄存器实现对POE内部RAM的1bit ecc错误注入和2bit ecc错误注入。

当对应的RAM有读操作发生时才会产生ECC错误注入，并上报对应的ECC错误中断，ECC错误注入寄存器中对应bit的值会被POE自动清掉，如果需要再一次ECC错误注入，需要重新配置ECC错误注入寄存器。

3.8.2.4 ECC 校验功能的打开关闭

POE支持软件通过配置POE模块级全局寄存器POE_ECC_BYPASS_CFG来控制POE模块中所有ECC_CHECK功能的打开或关闭。



3.8.2.5 中断注入

- 通过POE_PF_COM_INT_SET寄存器配置POE中的PF的中断对应bit进行中断注入。
- 通过POE_INT_SET寄存器配置POE中制式中断对应bit进行中断注入，每种制式都有一组POE_INT_SET中断注入寄存器。
- 通过POE_VCPU_INT_SET寄存器配置POE中CPU中断对应bit进行中断注入，每种制式都有一组POE_VCPU_INT_SET中断注入寄存器。

3.8.2.6 时戳功能

在Debug场景下，POE支持任务填充时戳，能够统计包任务/消息任务在POE内的latency，时戳精度为可配置，默认为0.1μs，宽度为24bit。

通过配置寄存器POE_TIMER_INSERT开启POE的打戳功能，POE根据寄存器POE_TIMERSTAMP_TYPE配置进行打戳，支持入口、出口及入口和出口之差的时戳填充；

对于256任务，若时间戳TLV存在，必须在倒数第二个TLV位置WORD6；并且在时戳功能开启后所有任务数据结构中必须携带该TLV。

其中在支持填充入口和出口之差的时戳时，支持统计一定时间间隔内包任务在POE内的时延最大值和最小值，通过寄存器POE_PKT_LAY_MAX/POE_PKT_LAY_MIN观测包任务在POE内的时延最小值和最大值；

支持软件通过寄存器POE_TS_LOAD_EN和POE_TS_LOAD_DATA向POE加载时戳，加载的时戳精度要求必须与POE_TIMER_CFG[timer_degree]保持一致。

须知

开启POE的打戳功能后，若没有Telemetry时戳，POE固定按照POE_TIMERSTAMP_TYPE填充时间戳。

3.9 注意事项

3.9.1 SWITCH_NULL 命令的使用注意事项

该产品应用代码中，CPU在发一个GET_POE/GET_GROUP之前会先发一个SWITCH_NULL操作，需要注意：

须知

同一个CPU发SWITCH_NULL和GET_POE/GET_GROUP之间必须加DMB指令，并且DISPATCH的POE_CPU端口配置为64Byte读写保序窗口。以此来保证SWITCH_NULL命令响应从POE输出后GET_POE/GET_GROUP才进入POE。否则，可能造成不可恢复的异常，可能会造成断流。



4 PPE

4.1 关键术语

如表4-1的术语将在本章节表示特殊的含义。

表 4-1 PPE 关键术语

术语	描述
PPE	Packet Process Engine，报文处理引擎。
RoCE	Remote DMA over converged Ethernet，建立在ETH上的远程DMA，是由IBTA标准化组织定义的一种在以太网上采用RDMA技术的网络互联技术。
BMU	Buffer Management Unit，包缓存管理单元。
POE	Packet Order Enforcer，包保序。
GE	兼容10M/100M/1G三种速率的MAC控制器。
XGE	10G MAC控制器。
TM	流量管理模块（ Traffic Management ），进行QoS处理，队列管理和调度流量整形等。
BD	Buffer descriptor，描述符，包括发送描述符和接收描述符，描述报文在主存中的存储信息，包括报文起始地址，报文解析结果等等。
ETS	Enhanced Transmission Selection，加强的发送数据选择。
TC	Traffic Class，DCB中优先级的8个优先级。
QoS	Quality of service，服务质量。 包描述符中一个域，用来表示包的优先级，在POE中QoS有8个优先级。
VQI	Virtual Queue Index，虚拟队列索引，POE使用VQI查表得到其内部的Group值。
Tag	每条流分配不同的Tag号，用来进行保序。



术语	描述
Pool	资源池，缓存空闲Buffer空间的指针。
PTP	Precision Time Protocol，即通常说的1588同步报文。
SP	Strict Priority，严格优先级调度。SP调度按照队列优先级的高低进行调度：高优先级队列先调度，低优先级队列后调度。
FW_TNL	Forward Tunnel，分发通道。PF配置分发到哪个FW_TNL，VF配置FW_TNL的具体内容。举例：PF配置哪个MAC地址映射到那个FW_TNL，VF配置这个分发通道的VQI是啥，QoS如何生成，等等。
接收解析描述符	接收报文的解析结果和存储在DDR中的位置等信息。
工作描述符	写给POE的简化的接收解析描述符。
报文发送描述符	发送报文存储在DDR中的位置和报文的处理方式等信息。
五元组	目的IP、源IP、3层协议号、目的端口号、源端口号。
三元组	目的IP、源IP、3层协议号。
流标号	Flow Label，IPv6协议定义的流标号。

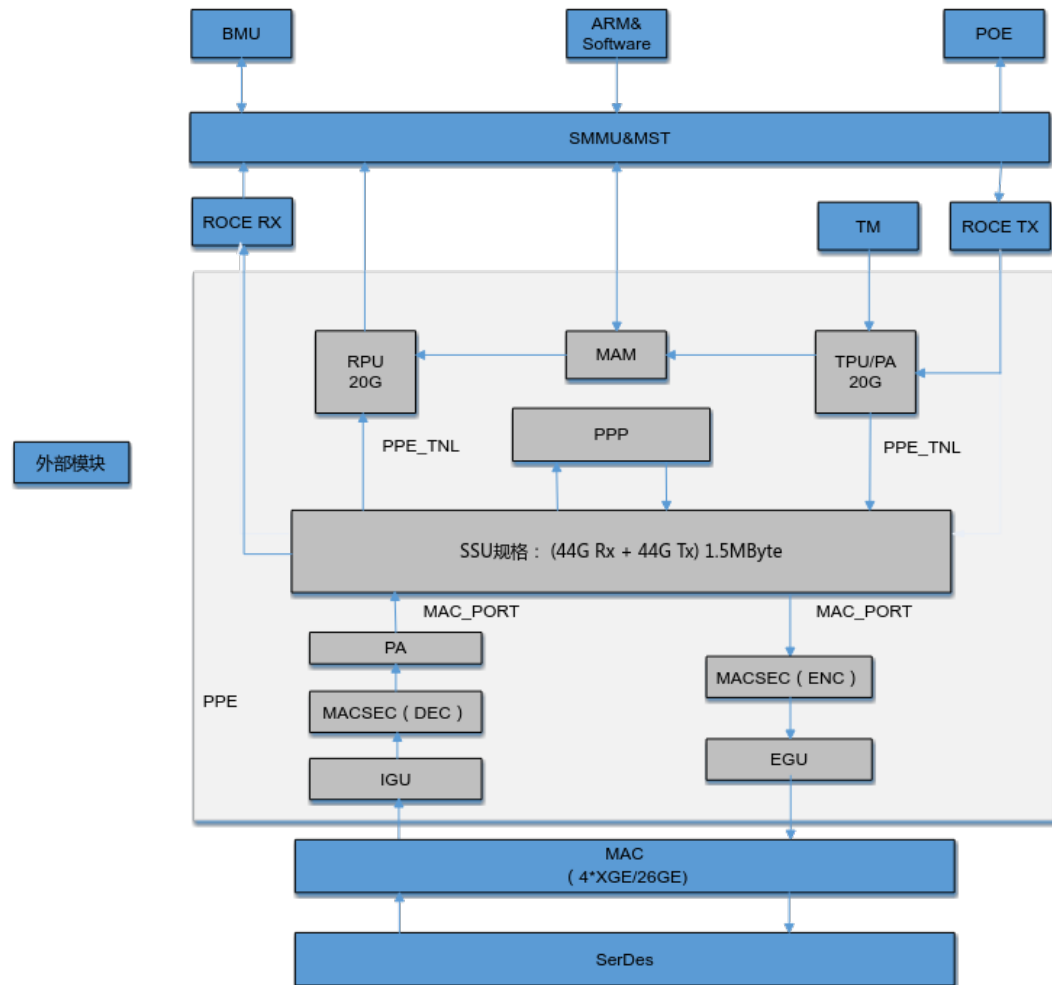
4.2 概述

包处理引擎（PPE）是控制MAC网口收发报文的模块，预申请缓存并管理。接收方向对报文进行解析产生调度信息、解析结果，并将其发送给其他加速模块或者软件使用；发送方向根据报文描述符对报文进行处理并通过网口发送出去。

各模块功能：

- TPU：Trasmit Process Unit，主要完成发送DMA功能，计算发送报文checksum等。
- RPU：Receive Process Unit，主要完成接收DMA功能，计算Tag、QoS等。
- PPP：Programmable Packet Processor，完成LSW查表和去CPU报文的分发信息查表。
- SSU：Storge Switch Unit，用于完成收发报文的存储、报文的TC队列管理，镜像、多播复制，使用PPP的转发查表结果完成报文的转发功能，同时基于各端口使用内部Buffer的空间控制生成Pause流控指示。
- EGU：Egress Unit，完成报文分发给各MAC。
- IGU：Igress Unit，完成MAC接收报文汇聚。
- PA：Packet Analysis，完成报文的解析和key信息的提取等。

图 4-1 PPE 结构图



4.3 特性

PPE有如下特性：

- 支持多种速率MAC对接。（IGU/EGU）
 - 最多支持4个XGE接口。
 - 最多支持4个10GE（BASE-R/BASE-R with BR FEC）。
 - 最多支持26个GE接口（Hi309a可用的serdes lane只有10个）；不支持半双工模式；1000BASE-X/SGMII支持提频到3.125Gbps。
 - 支持6路QSGMII与2个XGE并发。
 - 最大支持4路5GBASE-R/ 5GBASE-KR。
 - 最大支持9728Byte报文收发（逻辑需要留余量，最大长度可配，至少支持10240Byte）。
 - MAC接收最小支持报文长度是64Byte（含CRC）。
 - CPU发送最小长度15Byte（不含CRC），PPE发送方向padding到60Byte（不含CRC）。



- 支持接收侧MAC地址过滤，每个网口最多支持36个不带掩码的MAC地址和8个带掩码的MAC地址，所有网口共用配置表。注意：对端发送临终遗言需要配置白名单不过滤。
- 支持如下报文的识别和校验：（PA）
 - 支持IPv4、IPv6、TCP、UDP、GTPU、GRE、BOOTP、DHCP、SCTP、ICMP、802.1x、Q922、以太网OAM、ARP、ARP、IPSEC、BPDU报文，IP分片报文的报文解析、1588V2报文。
 - IPv6支持Fragment、ESP、AH、Hop-by-Hop Options、Routing、Destination Options、Mobility、7种扩展头。
 - 支持对已知格式的IPv4头、非分片TCP、非分片UDP checksum检查。
 - 支持对EtherType、IP-protocol、UDP/TCP/SCTP-Dport配置识别自定义报文，各支持15种报文类型（含协议报文和自定义报文），同一接口支持自定义和标准封装混合接收并转发。
 - 支持RoCEE V2报文识别，识别RoCEE V2报文的UDP端口可配，接收方向支持基于MAC口使能是否识别RoCEE V2。
 - 接收侧支持CAPWAP报文识别，PPE仅支持明文的识别。
 - PPE支持识别 控制明文、数据明文keep_alive报文、数据明文非keep_alive报文。
 - 对于控制密文、明文、数据明文keep_alive报文，仅支持识别报文类型。
 - 对于数据明文非keep_alive报文，支持按照隧道内层的规则提取各元组，提取的元组信息只用于进行hash得到fw_tnl（仅单模场景下支持）；隧道外L4 length 固定填UDP header length（8B）+capwap header length（由capwap header内提取的length与域段）。
 - 不支持隧道内层802.3snap识别，隧道内层如果是802.3snap封装时，PPE将无法识别隧道内L3报文头信息，PPE会将此类报文识别成L3类型是Unknow，统一按照L3 Unknow进行分发处理。
 - 不支持隧道内层MPLS识别，隧道内层如果有MPLS标签时，PPE将无法识别MPLS标签，会将此类报文识别成L3类型是Unknow，统一按照L3 Unknow进行分发处理。
 - 不支持提取隧道内层的自定义10bit/6bit信息。
 - 支持网口线速报文解析，并根据解析结果和报文类型完成分发。
 - 支持对报文有效性校验加速、畸形报文检查。
 - 支持提取4B DSA头，并基于4B DSA头进行分发。
 - 在同一接口同时支持标准封装和自定义封装报文混合处理。
- 支持接收侧MAC地址过滤，每个网口最多支持36个不带掩码的MAC地址和8个带掩码的MAC地址，所有网口共用配置表（IGU实现）。
- 支持多种生成分发通道FW_TNL（按以下规格生成，每条规则的优先级从高到低）：（PA/TPU完成报文各域提取，PPP进行优先级查表输出FW_TNL和目的MAC/CPU端口，基于FW_TNL表项部分在PPP，其他在MAM）
 - 支持256个TCAM分发表项，宽度320bit（也可使用512*160bit模式），以128个表项为1个bank，支持两级查表（可以根据L2或L3关键字查找），每级表的最小粒度为128个表项。



- PF统一维护上述表项，表项action：1）丢弃报文；2）进行下一级分发；3）得到分发结果；PF还需要配置FW_TNL和VFID映射关系，支持16组FW_TNL，每组有16个FW_TNL，每组可以单独设置VFID。
- EtherType自定义的报文，使用指定的模板提取8bit查表生成FW_TNL。
- 使用IP-protocol或UDP/TCP /SCTP -DportNum自定义报文，如果是其他自定义报文使能了提取模板，使用指定的模板提取8bit查表生成FW_TNL。
- 单模场景下，支持对不同类型报文根据不同元组进行hash得到FW_TNL。
- 支持基于FW_TNL映射出VQI值（支持使用配置的VQI+QoS作为最后的VQI值）、location、QoS的生成方式、TagType值、Tag的生成方式、PoolID、QoS不足丢弃使能。
- 支持多种QoS的生成方式：（PA/TPU完成报文各域提取，PPP进行优先级查表输出QoS）¹
 - 基于MAC分发，选择PRI查表生成QoS或使用FW_TNL配置的QoS。
 - 基于IP三/五元组分发，选择PRI或DSCP查表生成QoS或使用FW_TNL配置的QoS。
 - 基于自定义类型分发，使用自定义头里提取的6bit查表生成QoS或使用FW_TNL配置的QoS。
 - 以上查表的表项都在PF里面配置，每个VF内配置PF-QoS到VF-QoS的映射表。

说明

在配置FW_TNL的qos选择时，只能根据协议层选择本层和本层以下的QoS生成方式。

- 支持多种生成Tag的方式：（PA/TPU生成报文类型，PPP查表后生成Tag生成方式，RPU按照类型和生成方式计算得到Tag值）
 - 在分发结果中选择Tag的生成方式。
 - Tag[23:16]基于通道可配，低16bit使用SMAC和以太网类型值CRC-16作为Tag。
 - 协议解析模式（参加计算的各元素，可以基于通道掩码，（Tag[23:16]基于通道可配，低16bit使用如下方式产生）：
 - 第一类：IPv4封装的报文：
 - 非分片和首分片GTPU：使用五元组和隧道终点TEID进行CRC-16计算得到Tag。
 - 非分片和首分片UDP（除GTPU报文）/TCP：使用五元组进行CRC-16计算得到Tag（对PTP报文统一使用目的端口号为0x140，以保证不乱序）。
 - 非分片和首分片IPsec报文：目的IP/源IP/3层协议号、SPI进行CRC-16计算得到Tag。
 - 非首分片包和其他报文：使用三元组进行CRC-16计算得到Tag。
 - 第二类：IPv6封装的报文：
 - 非分片和首分片GTPU：使用五元组和流标号和隧道终点TEID进行CRC-16计算得到Tag。
 - 非分片和首分片UDP（除GTPU报文）/TCP：使用五元组和流标号进行CRC-16计算得到Tag（对PTP报文统一使用目的端口号为0x140，以保证不乱序）。



- 非分片和首分片IPsec报文：目的IP、源IP、3层协议号、SPI和流标号进行CRC-16计算得到Tag。
- 非首分片包和其他报文：使用三元组和流标号进行CRC-16计算得到Tag。
- 第三类：非IP报文，取SMAC和以太网类型值CRC-16作为Tag。
 - 自定义模板0：使用EtherType后指定offset（最大16个word）起连续16Byte中任选最多24bit作为Tag值。
 - 自定义模板1：仅模板可以单独配置。
 - 自定义模板2：仅模板可以单独配置。
 - 自定义模板3：仅模板可以单独配置。（所有端口共享4个自定义模板）
- 支持按分发结果（在分发结果内直接配置Tag Type值）生成Tag Type。（PPP得到FW_TNL，MAM基于FW_TNL返回Tag Type）
- 接收公共：（由MAM基于FW_TNL返回值，RPU按照返回值操作，其他由RPU完成）
 - 支持报文写入Cache的数据长度以CacheLine为粒度可配（即可以配置接收解析描述符和报文的前N个CacheLine为写操作是可Allocate的，N最大配置为31，31表示全报文进cache，这里的Cacheline大小是描述的软件看到的大小，定义是64Byte）。
 - 访问DDR时，访问总线时User域里面的type值寄存器可配。
 - 支持通过链表的方式对接收报文分离存放在主存中。
 - 支持小于等于64ByteIP报文的padding剥离，基于PPE_TNL配置使能。
- 接收存放报文：（PA/TPU输出长度，PPP比较得到需要类型，MAM根据Buffer情况得到最合适类型，其他由MAM完成）
 - 接收有16个FIFO用于存放向对应Pool申请的Buffer，每个FIFO和VF的逻辑Pool的映射由软件管理（该配置由底软或者平台软件完成），芯片对寄存器的访问按照配置表路由到对应VF的Pool上（即上层软件操作时不可见BMU的逻辑FIFO，只可见VF的Pool）。
 - 16个FIFO支持由软件直接分配的Buffer，应用于调试网口。
 - 16个FIFO可以单独配置为LIFO模式。
 - 支持基于FW_TNL里配置使用哪些Pool（每个分发表项可以设置使用A/B/C/D四种类型Buffer，每种类型可以单独使能，如果需要使用的Buffer类型未使能，直接丢弃该报文）。
 - 支持接收报文使用报文协议域指示的大小（如果是IP封装的报文：通过协议域total length和报文头长度计算得到报文总长度；其他报文取可配置的默认Buffer）查软件基于VF配置的水线表得到应该使用A/B/C/D的类型Buffer，然后选用不同类型的Buffer存放报文（最合适长度Buffer为空时向上或向下取合适的Buffer Size进行存储）。
 - 支持描述符和报文使用不同的Buffer存放。
 - 基于Pool的缓存不足时按照QoS（8级）丢弃报文（解析出报文的FW_TNL后找到对应的Pool号，查看该Pool在当前BMU的剩余缓存，通过接收报文的QoS决定是否丢弃报文）。
 - 存放报文支持基于Pool配置SKIP区（SKIP最大配置为15个CacheLine，链表存放的首片和其他片可以独立配置），支持基于通道在CacheLine对齐地址的后面空闲0~63个字节存放报文内容（只可配置首片）。



须知

PPE中某个Pool中Buffer数量低于水线时会向BMU申请Buffer，但若BMU中对应Pool频繁为空，则会频繁返回no_buf，浪费总线带宽。可以合理配置MAM_CFG_RX_BUFF_REQ_TIME来减少申请Buffer失败后的对应Pool的申请次数，节省总线带宽。

- 支持接收方向收到的报文添加到POE，基于PPE_TNL配置去POE消息中的SourcePort。
- 发送公共：（TPU完成）
 - 支持对链表方式存放的报文进行读取发送，描述符和报文支持分离存放。
 - 发送方向支持软件指定报文的TCP、UDP的Checksum计算（最大支持9600Byte报文计算Checksum，IPv4支持扩展头，IPv6不支持扩展头），对于IPv4头硬件固定计算。
 - 支持报文发送完成后自动释放对应Buffer。
 - 支持软件通过报文发送描述符配置发送报文的读操作是否为Read invalid和Read Allocate。
 - 访问DDR时，访问总线的user中的type寄存器可配。
 - 发送支持32个CPU独立推包寄存器，以4个一组的粒度分配给各VF使用，并基于推包接口查询推送是否成功。
- 发送Y.1731功能：（TPU完成）
 - 发送支持软件指定的32条流统计。
 - 发送支持软件对指定报文将指定流统计的值添加到报文的指定位置。
 - 发送支持软件对指定报文将当前RTC的时间值（PTP本地时间）添加到报文的指定位置。
- 支持反压CPU Station功能（逻辑线反压）：（OOO输出读写延时，TOP基于延时生成反压）
 - 基于访问总线的平均延时值和配置阈值反压或解反压总线。
 - 支持连续反压时间值可配，连续反压CPU Station超时后，强制结束反压。
- 支持流控功能（SSU完成）
 - 支持基于内部缓冲mem的使用状态进行流控。
 - 支持基于BMU的Buffer的使用状态进行流控。
 - 支持根据软件指示进行流控。
 - 支持超时功能，停止收到报文达到流控反压时间达到门限后退出流控，直到再次收到报文或流控状态解除后，才会重新触发流控。
 - 支持端口级流控水线单独可配。
- 地址处理（MAM/TPU/RPU）
 - 支持48位虚拟地址访问DDR和其他加速器。
 - 支持加速器互访配置为PA地址访问。
 - 支持加速器间访问128bit地址对齐。
- 大小端处理：（TPU/RPU）
 - CPU直接分配和读出Buffer的寄存器接口支持大小端基于VF可配，其他寄存器接口只支持小端，该接口位宽超过32bit时，软件需要发64bit操作。



- 加速器间互访只支持小端；CPU推送发送描述符地址只支持小端。
- 读写报文时只支持字节流大端，读写描述符支持基于PPE_TNL大小端可配（32bit、64bit可配，默认32bit大端）。
- 对应超过32bit统计，软件需要先读低32bit，然后再读高32bit。
- 支持多模隔离和虚拟化：（TPU/RPU/TOP）
 - 支持5个VF制式。
 - 寄存器空间分为PF和VF，每个VF的寄存器一致，即每个VF里面同样功能寄存器相对该VF的起始位置相同。
 - PF可以访问所有寄存器。
 - 每个VF有1M空间，前1M~64KB是PF在VF中的只读映射，最后64KB是VF自己的功能空间。
 - 单模复位：全局可以配置哪些VFID处于disable状态：
 - 接收方向：收到处于disable状态的VFID报文直接丢弃。
 - 发送方向：不发送处于disable状态VFID的报文（此时的缓存直接丢弃由BMU重新初始化该Pool）。
 - 资源回收：升级或单模disable时可以将某Pool本地申请关闭，软件可以直接读出指定Pool的Buffer。
 - 各VF中断独立上报。
 - 基于VF单独配置STREMIID。
 - 每个VF支持配置使能和工作使能寄存器。配置使能无效时，软件对该VF的配置操作被硬件丢弃；工作使能有效后，模块才能开始工作，如开始预申请Buffer、对总线发送队列状态等。（VF工作使能寄存器要放在VF寄存器域段内）
- Telemetry特性
 - PPE支持对接收方向记录入口时间写入接收描述符。
 - PPE支持接收方向携带INT/IOAM info的报文解析，支持解析格式为IPv6、GRE、TCP/UDP、Vxlan-gpe。
 - PPE支持发送方向根据描述符中INT/IOAM的使能和对应信息，对发送除需要TSO以外的报文进行发送时间和接收时间的差值计算，并将差值和发送时间放入报文制定位置，重新计算checksum后，将报文进行发送。
 - PPE支持的统一时间戳格式为{32bit秒信息，2bit为0，30bit纳秒信息}。
 - INT/IOAM头不会作为PPP查表关键字进行流表和分发表查询。
- 支持LAN Switch在线
 - 支持计算子系统异常挂起时，网口LSW功能正常转发，HOST侧报文丢弃。
- 支持临终遗言功能，软件可配置source addr，code和flag。
- 支持XSEC读写安全访问。PPE根据总线user中安全指示位，决定是否可以对XSEC中的秘钥进行读写。
- 性能：数据不包含帧间隙和物理层开销。
 - HOST侧收发报文长度在330B以上可以达到20Gbps（相当于7.6Mpps），基于平均延时300ns。
 - MAC侧收发报文长度在60B以上任意包长可以达到44Gbps线速（相当于65.5Mpps）。
- 带宽



- 无线性能要求：支持PPE最大1518Bytes @20Gbps收发规格。
- 网络性能要求：
 - MAC LSW支持44Gbps 64B线速。
 - AR转发性能 10Gbps@330B/6.5Mpps@64B。
 - 防火墙传统转发性能 5Gbps@256B/4.5Mpps@64B。
 - AC转发性能20Gbps@512B/5Mpps@64B。
- 当接收方向突发小包时（比如60Byte），在SSU没有达到流控水位前可以线速接收，达到流控水位后，如果对端不响应流控会出现丢包，突发接收小包时，发送方向带宽会被接收小包占据，导致发送性能下降。

须知

这里描述的MAC带宽是各种MAC速率加起来的最大物理带宽，不是平均带宽。比如MAC支持44Gbps流量，是指所有的MAC支持的最大速率和是44Gbps。

下述特性是LSW特性（LSW由PPP/SSU配合完成）

VLAN特性：

- 支持端口属性配置为access/hybrid/ trunk。
- 支持基于物理端口VLAN TAG处理，支持删除或添加一层或两层VLAN TAG。
- 支持QinQ模式，QinQ报文可以支持处理两层VLAN。
- 入口流表和出口流表的表项结果支持报文的优先级修改，即支持修改VLAN 的PRI域；入口流表和出口流表也支持内层和外层VLAN TAG的添加，删除和修改等操作。

LSW转发特性：

- 支持MAC侧26个端口，CPU侧支持最大26个PPE_TNL。
- 支持基本的L2 LAN Switch，基于L2表MAC、VLANID的报文交换，L2表最大支持6K深度和128个逃生桶，所有端口共享；对于QinQ报文，支持选择2层VLAN个中的任何一个做L2转发。
- 支持硬件或软件的L2表项的MAC地址自学习。硬件自学习支持对MAC和CPU侧学习。软件自学习通过硬件上报中断知会软件完成；支持基于端口设置由硬件还是软件进行自学习或关闭自学习。
- 支持基于端口的学习MAC数量限制。
 - 支持MAC学习达到上限后，超过上限的MAC报文丢弃。
- 支持L2表中每个表项可以单独设置是否老化。
- 支持源MAC命中丢弃。
- 支持MAC地址漂移，对于非静态漂移的MAC地址进行可以配置学习，并上报中断，对于静态MAC地址的漂移报文进行可配置丢弃并上报中断。
- 支持基于MAC侧端口和CPU侧PPE_TNL配置成直通模式。配置成直通模式的端口和PPE_TNL会bypass流表，L2表分发，Vlan编辑和过滤，可以直接配置转发端口或PPE_TNL。



- 支持发送侧报文进行LSW直通转发。发送侧报文由软件在发送描述符中指定转发端口或PPE_TNL，此类报文在查表时会bypass流表，L2表分发，Vlan编辑和过滤，按照软件要求直接转发。
- 支持PPE_TNL之间通讯：支持通过流表，L2表或者根据发送描述符中的LSW直通指示，实现发送报文的直接环回接收。

以下是多播/广播/洪泛特性：

- 支持多播复制，在L2表或者入口流表表项里面可以指定分发类型为多播和多播叶子表的表项ID，多播叶子表的表项最大支持1024个。
- 支持对多播/广播/未知报文风暴的抑制，当内部的多播/广播/未知报文积累到水线时，对新收到的多播/广播/未知报文进行尾丢弃。
- 对未知单播和多播，基于端口配置支持在VLAN域内进行洪泛。
- 对于硬Rocee的报文不支持多播与镜像。

以下是报文过滤特性：

- 支持基于物理端口的VLAN过滤特性，支持一张4096深度的VLAN表，支持入口和出口基于VLAN ID的报文过滤。
 - 入口VLAN过滤在入口流表之后进行。
- 支持基于流表的ACL过滤功能，可以基于二层报文的MAC/VLAN/ETH Type、三/四/五层IPv4/IPv6/UDP/TCP/SCTP等报文的三元组/五元组、自定义协议类型字段进行报文过滤。
- 支持在入口和出口流表进行ACL处理。入口和出口流表共享1536*160b的表项，可以以128*320b为粒度，进行分配。
- 支持端口单播、多播、广播和洪泛隔离。

以下是LSW其他特性：

- 支持入口和出口镜像功能，入口支持基于入端口，入口流表的镜像；出口支持基于出端口和出口流表镜像，镜像处理将一个端口收到的所有报文复制到CPU或另一个端口，入口镜像和出口镜像的目的端口各自一个。
- 支持对外的MAC口有2bit生成树控制：0b00：正常转发报文；0b01：该端口收到的报文全部丢弃；0b10：仅BPDU报文不丢弃，其他报文全部丢弃，学习；0b11：仅BPDU报文不丢弃，其他报文全部丢弃，不学习；生成树支持STP、RSTP，支持MSTP，最大支持16个实例。
- 支持最多4个trunk组，每个trunk组最多8个成员，使用MAC地址或IP地址hash选路，不支持主从备份，即硬件不支持自动切换trunk组成员，由软件处理。
- 内部包缓存最大1.5MB。
- 每个MAC发送端口和CPU接收队列最多支持4个优先级，支持MAC发送端口队列和CPU的接收队列基于端口和优先级设置水线丢弃；支持端口和PPE_TNL队列出队时按照Shapping与SP+WRR/DRR调度。
- 支持BPDU报文透传，也支持在流表设置过滤BPDU。
- 入口支持20个CAR过滤，分别是1个全局性的CAR过滤，基于DMAC的单播，多播，广播各1个全局性CAR过滤，8个基于单播优先级的CAR过滤和8个基于多播优先级的CAR过滤；在SSU中实现。
- 入口支持基于入口流表的32个CAR过滤；在PPP中实现。
- 出口支持对未知单播、未知多播，单播和多播、广播的CAR过滤，共支持5个CAR；在PPP中实现。



须知

对于CAR的使用有以下几点说明：

1. CAR规格并不是针对产品业务承诺的规格，是预置的一个处理潜在的对付跳水的手段；如果出现跳水场景，可以尝试各种CAR方式，典型应该使用全局CAR或者低优先级的CAR。期盼使跳水性能有所恢复。
2. PPP不支持未知广播的CAR，如果需要对未知广播报文进行限速，可以使用SSU基于广播的全局CAR。
3. PPP中CAR操作采用优先选择原则：流表CAR优先级>未知单播、未知多播，单播和多播、广播的优先级；做过流表CAR的报文不再进行报文类型CAR运算处理。
4. SSU中CAR操作采用并行使用原则：即单播会同时进行单播CAR和优先级CAR运算，多播广播类似。

- 支持各物理口收发报文的统计。
- 支持转发报文的统计。
- 支持ECC错误注入。
- 支持记录第一个报文的key。
- 支持靠MAC侧，基于MAC粒度配置内环和外环。
- 所有丢包地方都有基于丢包类型的统计。
- 所有MEM都有ECC保护，分别记录1bit和2bit错误地址，记录第一次地址。
- 配置类TCAM支持背景扫描校验ECC，并单独上报中断状态。
- 支持配置类MEM/TCAM出现1bit ECC错误纠错和检错功能，由软件对该mem全部回读并重写。
- 支持动态门控，模块没有业务时，自动关闭时钟。

4.4 功能描述

4.4.1 相关模块

在系统中，PPE模块会和下面多个模块进行交互，分别描述如下：

- BMU：PPE向BMU申请Buffer用来存储接收的报文；向BMU释放已经发送出去的报文所占的Buffer。
- POE：PPE通过向POE添加接收报文的工作描述符，将报文交给下一个处理单元。
- TM/CPU core：TM/ CPU core将发送描述符的地址添加PPE的发送FIFO，PPE根据发送描述符，从DDR中回读报文，并进行报文发送。
- DDR：PPE接收存放和发送报文的存储模块。
- MAC：通过MAC完成网口链路层和数据层报文的收发，MAC包括GE/10GE。
- Rocee：发送方向PPE完成Rocee与NIC报文的汇聚，通过MAC发送出去；接收方向PPE完成Rocee与NIC报文的分发，将Rocee报文发送给Rocee模块。PPE不针对Rocee报文做相应的编辑。

4.4.2 PPE 通道和网口互联状况和使用场景



4.4.2.1 PPE_TNL 和 MAC_TNL

PPE支持26个通道，即26个PPE_TNL。

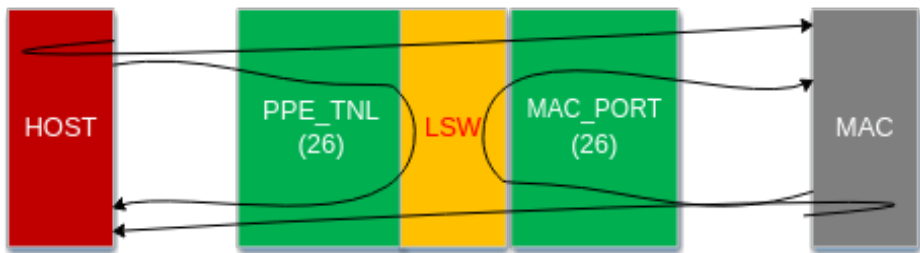
PPE支持和26个MAC对接。

发送方向，可以指定PPE_TNL和哪个MAC_PORT直通，直通时只能一对一；

接收方向，可以指定MAC_PORT和哪个PPE_TNL直通，直通时只能一对一。

非直通的PPE_TNL和MAC_PORT的报文，都会经过LSW进行转发。

图 4-2 PPE 通道使用场景



PPE支持26个网口，同一个网口同一时刻只能配置成为一种网口速率模式。为了保证性能和降低PPE LSW和直通表配置的复杂度，MAC网口和PPE TNL的初始关系如下图4-3所示。

图 4-3 MAC 组网场景

AXI Master	Port #	(FE/1000BASE-X/SGMII/2.5GBASE-X) 说明: 1000BASE-X/SGMII支持提到3.125Gbps	10GE (BASE-R/BASE-R with BR FEC) 或者 5GBASE-R/ 5GBASE-KR	QSGMII	RGMII/MII	PPE Tnl
Master 000	Port0	GE0	10GE2	QSGMII0		PPE Tnl0
	Port1	GE1	NotUsed	NotUsed		PPE Tnl1
	Port2	GE2	NotUsed	NotUsed		PPE Tnl2
	Port3	GE12	NotUsed	NotUsed	RGMII/MII	PPE Tnl3
	Port4	GE3	10GE3	QSGMII1		PPE Tnl4
	Port5	GE13	NotUsed	NotUsed		PPE Tnl5
	Port6	GE14	NotUsed	NotUsed		PPE Tnl6
	Port7	GE15	NotUsed	NotUsed		PPE Tnl7
	Port8	GE4		QSGMII2		PPE Tnl8
	Port9	GE16		NotUsed		PPE Tnl9
	Port10	GE17		NotUsed		PPE Tnl10
	Port11	GE18		NotUsed		PPE Tnl11
	Port12	GE5		QSGMII3		PPE Tnl12
	Port13	GE19		NotUsed		PPE Tnl13
	Port14	GE20		NotUsed		PPE Tnl14
	Port15	GE21		NotUsed		PPE Tnl15
	Port16	GE6		QSGMII4		PPE Tnl16
	Port17	GE22		NotUsed		PPE Tnl17
	Port18	GE23		NotUsed		PPE Tnl18
	Port19	GE24		NotUsed		PPE Tnl19
	Port20	GE7		QSGMII5		PPE Tnl20
	Port21	GE8		NotUsed		PPE Tnl21
	Port22	GE9		NotUsed		PPE Tnl22
	Port23	GE25		NotUsed		PPE Tnl23
	Port24	GE10	10GE0			PPE Tnl24
	Port25	GE11	10GE1			PPE Tnl25

MAG集成6个HiMACV110 X4版本和2个HiMACV110 X1版本，可以支持上述的网口速率模式；。

PPE网口和TNL的使用原则如下：



- 所有网口的速率之和小于等于44Gbps，否则可能会出现报文丢包。
- 单个PPE port只能适配一种MAC速率，“NotUsed”表示当使用某种速率模式时，其他对应的port无法工作。
- FE/1000BASE-X/SGMII/2.5GBASE-X接口模式，在同一个HiMACV110 X4版本中可以同时工作。
- 10G速率模式时，一条Serdes Lane会使用1个PPE Port。
- 5G速率模式时，一条Serdes Lane会使用1个PPE Port。
- 使用QSGMII速率模式时，一条Serdes Lane会使用4个PPE Port。
- PPE_TNL可以通过配置PPP的表项配置和任意网口对接；默认值为一一顺序对应。
- 最多支持4个AN_TRANING对应4个XGE；其他口不支持。
- 支持RGMII/MII调试网口，固定使用PORT3。

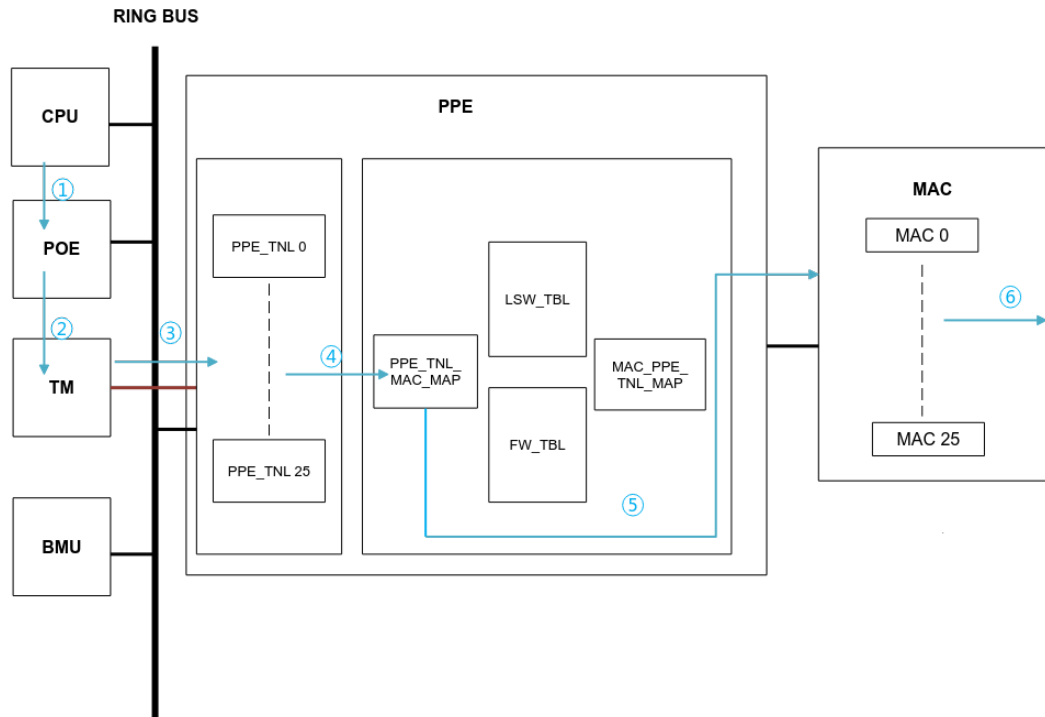
表 4-2 各速率与频点说明

主类	子类型	SerDes Lane 数目	SerDes 标准速率 (Gbps)	SerDes 扩展频点1 (Gbps)	SerDes 扩展频点2 (Gbps)	SerDes 并口位宽 (bit)	协议类型
FE	100BASE-FX	1	1.25	NA	NA	10	IEEE 802.3
GE	1000BASE-X	1	1.25	3.125	NA	10	IEEE 802.3
	SGMII(10M/100M/1000M)	1	1.25	3.125	NA	10	serial-GMII specification version 1.7
	QSGMII	1	1.25x4	NA	NA	10	QSGMII specification version 1.2
2.5 GE	2.5GBASE-X	1	3.125	NA	NA	10	IEEE 802.3cb
5GE	5GBASE-R/5GBASE-KR	1	5.15625	NA	NA	40	IEEE 802.3cb
10GE	10GBASE-R	1	10.3125	NA	NA	40	IEEE 802.3
	10GBASE-R (with BR FEC)	1	10.3125	NA	NA	40	IEEE 802.3

4.4.3 典型数据流

4.4.3.1 发送直通数据流

图 4-4 发送直通 MAC 示意图



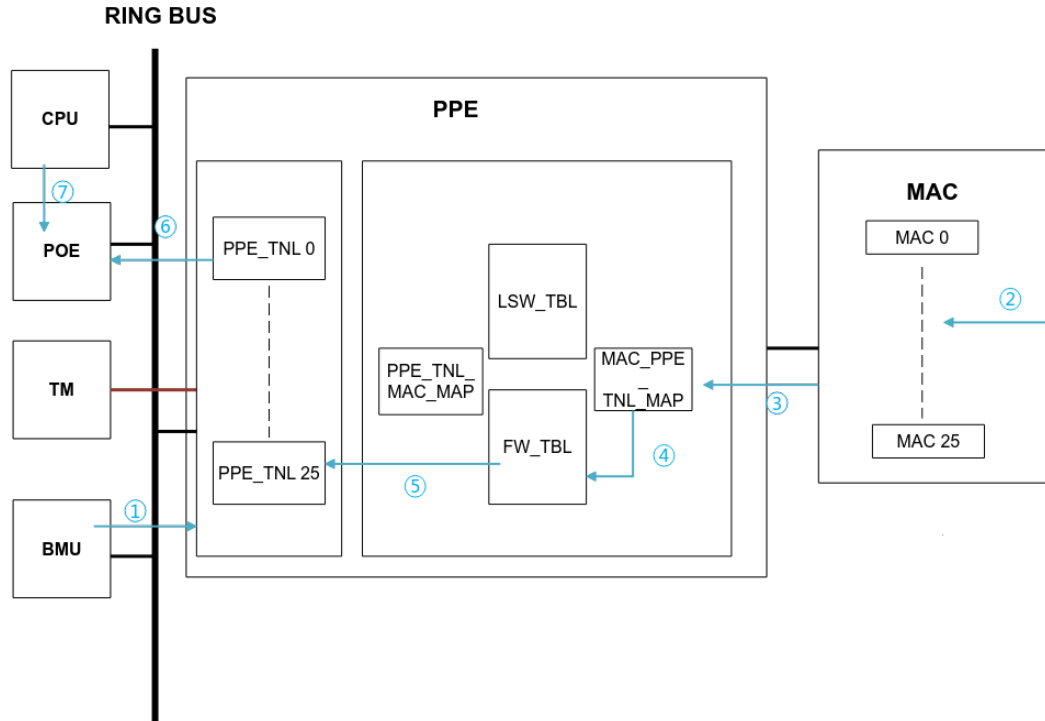
具体步骤如下：

- 步骤1** CPU CORE预先将待发送描述符的地址写入POE；
- 步骤2** POE将任务推送给TM；
- 步骤3** TM根据流量、QoS等配置规则，以及PPE发送配置缓存的空满情况，将发送描述符的地址发送给PPE（如果是通过TM推送才需要该步骤）；
- 步骤4** PPE根据发送描述符的地址取出发送描述符，按照发送描述符的发送报文信息，从缓存中读取待发送报文；
- 步骤5** PPE根据PPE_TNL和MAC的映射关系查询到该PPE_TNL直通的MAC通道，将报文发给对应的MAC通道；
- 步骤6** MAC将报文发送出去。

----结束

4.4.3.2 接收直通数据流

图 4-5 接收直通 CPU 示意图



步骤1 PPE预先从BMU申请缓存地址。

步骤2 MAC接收到报文。

步骤3 PPE根据MAC和PPE_TNL和映射关系查询到该直通的PPE_TNL。

步骤4 PPE通过报文头信息，查找分发表，得到去POE的分发信息。

步骤5 PPE将报文发送给对应的PPE_TNL。

步骤6 PPE将报文写入DDR，并将报文的消息推送给POE。

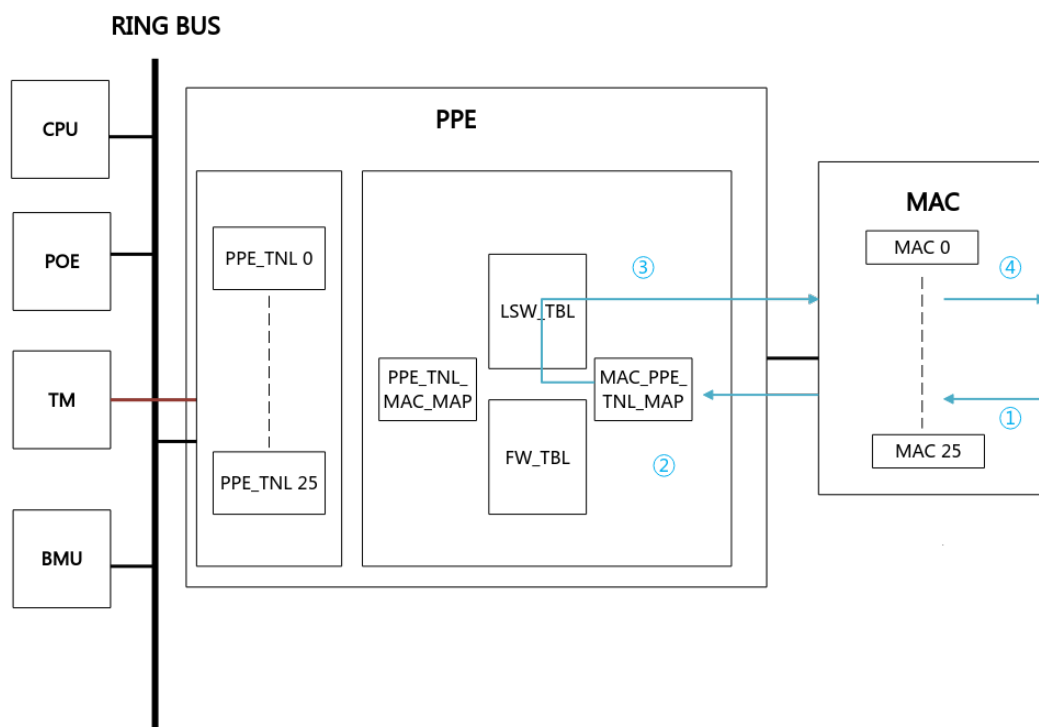
步骤7 CPU从POE中取出消息处理报文。

----结束

4.4.3.3 LSW 数据流

4.4.3.3.1 MAC 间转发

图 4-6 MAC 间转发示意图



步骤1 某个MAC收到报文。

步骤2 PPE根据MAC和PPE_TNL的映射关系查询到该MAC非直通CPU。

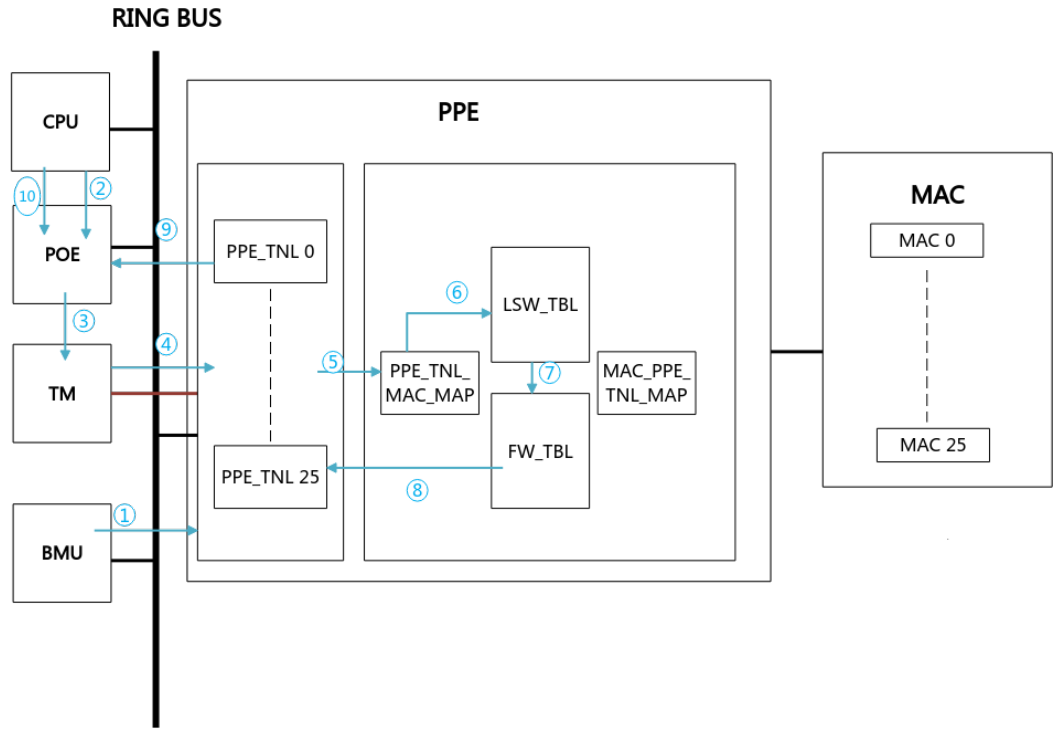
步骤3 PPE查询LSW_TBL，得到目的MAC。

步骤4 目的MAC将报文发送出去。

----结束

4.4.3.3.2 CPU 间数据流

图 4-7 CPU 到 CPU 示意图



步骤1 PPE预先从BMU申请缓存地址。

步骤2 CPU core预先将待发送描述符的地址写入POE。

步骤3 POE将任务推送给TM。

步骤4 TM根据流量、QoS等配置规则，以及PPE发送配置缓存的空满情况，将发送描述符的地址发送给PPE（如果是通过TM推送才需要该步骤）。

步骤5 PPE根据发送描述符的地址取出发送描述符，按照发送描述符的发送报文信息，从缓存中读取待发报文。

步骤6 PPE根据PPE_TNL和MAC的映射关系查询到该PPE_TNL非直通MAC。

步骤7 PPE查询LSW_TBL，得到PPE_TNL。

步骤8 PPE通过报文头信息，查找分发表，得到去POE的分发信息，PPE将报文发送给对应的PPE_TNL。

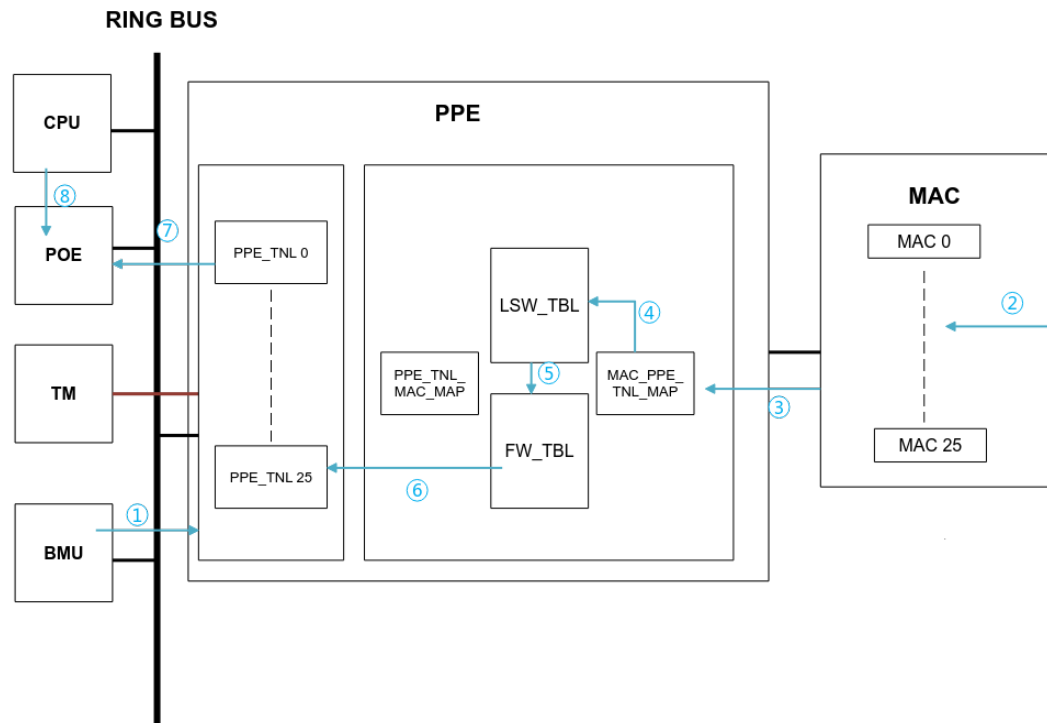
步骤9 PPE将报文写入DDR，并将报文的消息推送给POE。

步骤10 CPU从POE中取出消息处理报文。

----结束

4.4.3.3.3 MAC 到 CPU 数据流

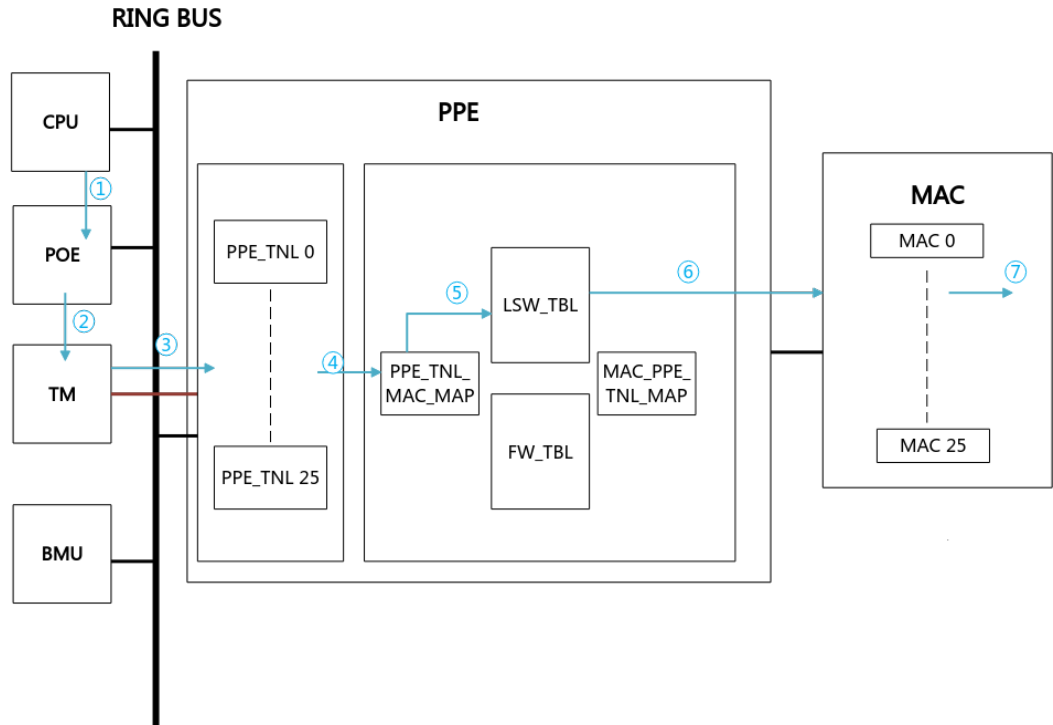
图 4-8 MAC 到 CPU 示意图



- 步骤1** PPE预先从BMU申请缓存地址。
- 步骤2** MAC接收到报文。
- 步骤3** PPE根据MAC和PPE_TNL和映射关系查询到该MAC非直通CPU。
- 步骤4** PPE查询LSW_TBL，得到PPE_TNL。
- 步骤5** PPE通过报文头信息，查找分发表，得到去POE的分发信息。
- 步骤6** PPE将报文发送给对应的PPE_TNL。
- 步骤7** PPE将报文写入DDR，并将报文的消息推送给POE。
- 步骤8** CPU从POE中取出消息处理报文。
- 结束

4.4.3.3.4 CPU 到 MAC 数据流

图 4-9 CPU 到 MAC 示意图



步骤1 CPU core预先将待发送描述符的地址写入POE。

步骤2 POE将任务推送给TM。

步骤3 TM根据流量、QoS等配置规则，以及PPE发送配置缓存的空满情况，将发送描述符的地址发送给PPE（如果是通过TM推送才需要该步骤）。

步骤4 PPE根据发送描述符的地址取出发送描述符，按照发送描述符的发送报文信息，从缓存中读取待发送报文。

步骤5 PPE根据PPE_TNL和MAC的映射关系查询到该PPE_TNL非直通MAC。

步骤6 PPE查询LSW_TBL，得到目的MAC，将报文发送给目的MAC。

步骤7 目的MAC将报文发送出去。

----结束

4.4.4 常用业务描述

4.4.4.1 调试网口功能

PPE的任意1个网口可以配置为调试网口，调试网口可以不使用TM、BMU、POE，CPU通过网口中断完成报文的收发。

发送方向首先配置调试网口功能使能和调试网口通道号PPE_TPU_TEST_PORT_CFG（0x6000），配置调试网口总线配置PPE_TPU_TEST_PORT_USER_STRMID



(0x6010)、PPE_TPU_TEST_PORT_SMMU_USER (0x6014)、PPE_TPU_TEST_PORT_SMMU_USER_SSET (0x6018)，然后CPU通过推包寄存器PPE_CORE_ADD_TX_DB_WORD0 (0x11E00+0x10*core_num)、PPE_CORE_ADD_TX_DB_WORD1 (0x11E04+0x10*core_num)直接将发送描述符地址推送到PPE，PPE完成发送后将OWN域修改为0x5，软件查询到为0x5后表示报文已经发送完，可以回收Buffer；

接收方向PPE通过配置调试网口使能寄存器RPU_TEST_PORT_CFG (0x0BC0)使能相应的测试通道，配置超时寄存器RPU_TEST_PORT_TIMEOUT (0x0520+tnl_num*0x40)、接收报文数量寄存器RPU_TEST_PKT_CNT_FG (0x0524+tnl_num*0x40)控制收到软件配置的报文数后上报中断，或收到报文后没达到配置报文数超时后也会触发中断，软件收到中断按照软件记录的Buffer地址直接去处理报文。PPE的调试网口的报文写DDR的outstanding需要为1，否则会出现多个写操作乱序，导致软件出现一致性问题。

4.4.4.2 RTC 功能

RTC是real time clock缩写，完成了一个实时时间功能。

PPE支持1个RTC：

RTC用于发送和接收1731时间戳功能、所有端口的1588时间戳功能。

4.4.4.3 1731 功能

接收和发送各支持32条1731流统计。

4.4.4.3.1 发送方向

软件通过描述符指定业务报文或1731报文所属的流，并指示硬件进行统计。对于1731报文，软件进一步通过描述符指定硬件在其上添加1588时间戳和填写流统计信息。在发送方向上，PPE对报文类型不敏感，因此软件需确保描述符中的1731报文处理指示准确无误。

4.4.4.4 1588 功能

4.4.4.4.1 1588 opcode

软件发送需要PPE/MAC处理的1588报文时，需要在DMAC前增加8Byte的1588操作符，该操作符用于MAC处理报文，MAC按照8Byte的操作符处理完报文后，会将8Byte的操作符剥离掉后发送出去。

软件通过发送描述符中的PTPM[0]指示是否是如下图4-10格式报文：



图 4-10 MAC 处理的 1588 报文格式

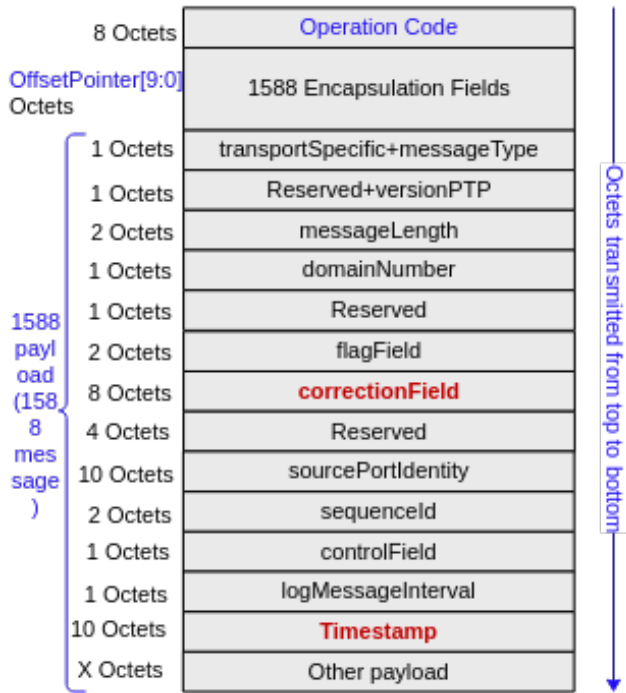


表 4-3 8Byte1588 操作符内容

Bit	Filed	usage
bit[63:61]	Opcode[2:0]	0b000: 不处理; 0b001: 打80bit时间戳; 0b010: 1588报文的原修正域值减去寄存器配置的64bit非对称延时后覆盖原修正域; 0b011: 将1588报文的原修正域值修正发送方向延时后加上申请到的47bit时间戳后覆盖原修正域; 其他: 保留。
bit[60]	ReturnTS	通过中断上报时间戳。
bit[59]	ReturnType	固定填0。 0b0: 32-bit is consist of 2bit S + 30bit NS both of which are from 80bit timestamp; S value is placed on MSB; 0b1: 32-bit is directly from the low 32bit of 48-bit timestamp。



Bit	Filed	usage
bit[58]	IPv6	IPv6/UDP packet。 软件在报文的最后2个Byte需要填充checksum修正域，MAC修改时间戳后会更新该域。
bit[57:48]	OffsetPointer[9:0]	报文DMAC到1588头的偏移值，必须是偶数字节。
bit[47:24]	SequenceInfo[23:0]	上报时间戳的伴随信息。 bit[23:16]：软件填充的端口号； bit[15:0]：软件填充的1588报文的sequenceid。
bit[23:0]	Rsv[23:0]	保留。

在发送1588报文时，在配置表项需要注意和保证不对1588报文进行报文编辑处理：

- 报文走直通模式时，转发模块不对1588报文进行任何处理。
- 报文走LSW模式时，建议软件可以采用以下三种方式：
 - 通过LSW直通报文格式（在BD描述符的SC域置为1），这样LSW不对1588报文进行任何VLAN编辑，直接发到对应的MAC端口；
 - LSW入口有VLAN编辑，可通过LSW的入口流表设置表项对1588报文入口的VLAN编辑进行旁路处理，出口和出口流表不设置任何VLAN编辑；
 - LSW没有任何VLAN编辑的应用场景下，不需要对1588报文进行特殊处理。

4.4.4.4.2 接收方向处理

每个报文在MAC传输到PPE时，会携带{2bit秒，30bit纳秒}的时戳信息，PPE在结合本地RTC的时间信息进行计算后，恢复成64bit时间戳形式：{32bit恢复后秒信息，2b原始秒信息，30b原始纳秒}。

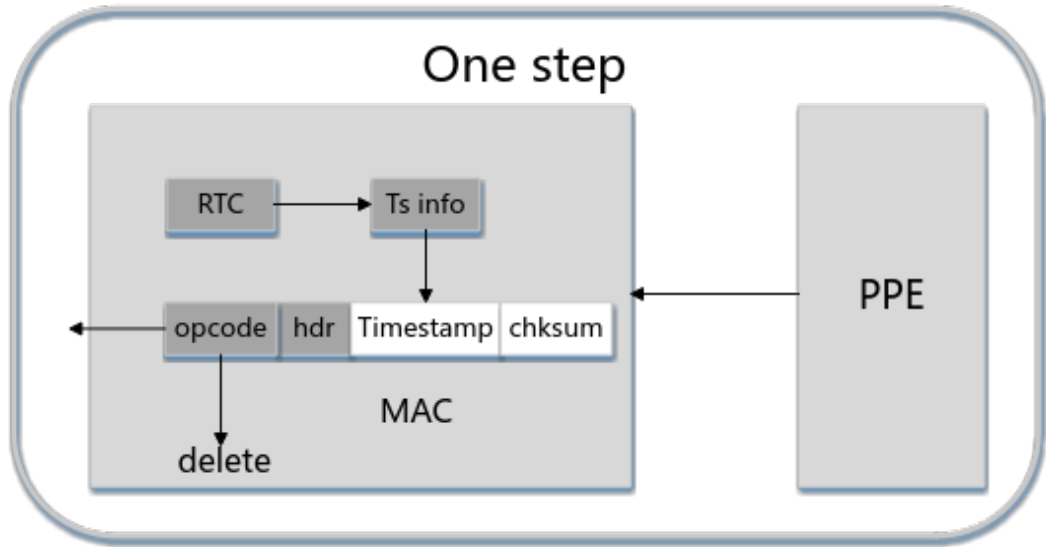
在POE模式下，硬件对所有非分片报文都通过接收描述符上报接收时间戳，软件自己按照需要取时间戳进行处理。

4.4.4.4.3 发送方向处理

在发送1588报文的过程中，当PPE接收到报文描述符并确认该报文为1588报文时，会直接跳过前8Byte的操作符，然后开始解析报文内容。在转发过程中，为了确保报文操作符中的OffsetPointer信息准确无误，软件必须避免对1588报文进行任何形式的修改。

对于1588报文在发送方向上的时戳处理，主要有两种方式：单步法和两步法，如下图4-11所示。

图 4-11 1588 报文发送单步法

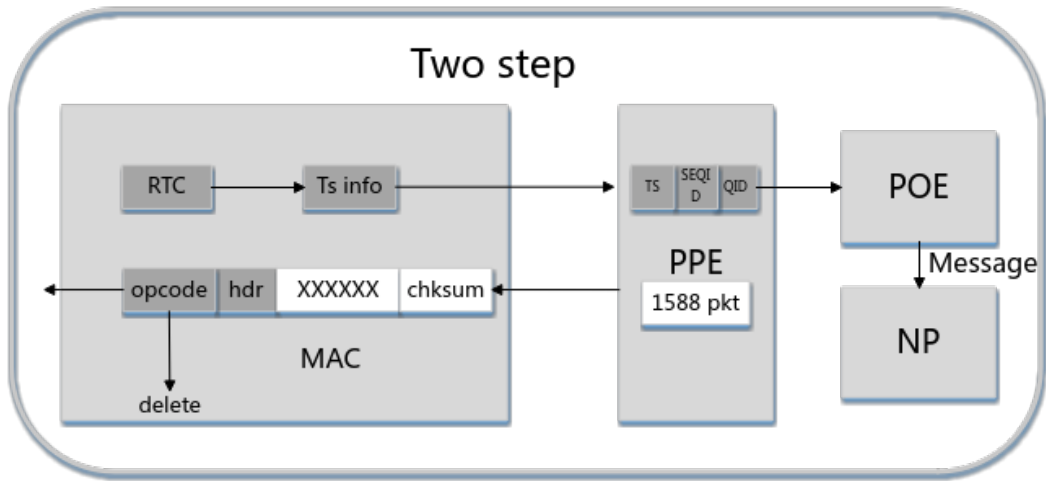


单步法流程如下：

- 步骤1** CPU/NP将1588操作符置于1588报文的开始部分，并将其传递给PPE。
- 步骤2** PPE随后将包含1588操作符的完整1588报文转发至MAC，并明确指示MAC是1588报文。
- 步骤3** MAC接收到1588报文后，根据操作符中的指令，利用从RTC获得的时戳信息进行相应的计算，并将结果填写到报文中指定的位置。之后，MAC会移除操作符并将处理后的报文发送出去。
- 步骤4** 对于 采用IPv6+UDP封装的1588报文，在报文末尾的两字节中需填充checksum修正域（此2Byte计算在UDP头中的len长度中）。当 MAC完成时间戳的更新后，需要重新计算checksum，并更新该域，确保接收端能够正确的计算出checksum。

----结束

图 4-12 1588 报文发送两步法





两步法流程如下：

步骤1 PPE发送1588报文。

PPE向MAC发送一个1588报文，该报文携带一个特殊的操作符，指示MAC需要进行时间戳回传；

步骤2 MAC处理并回传时间戳信息。

MAC收到1588报文后，根据操作符中的指示，将从RTC获取的时戳信息，经过计算后回传给MAG，传输信息为{8' b0, 8bit pid, 16bit seqinfo, 2bit秒, 30bit纳秒}；并剥离操作符后将报文发送出去，并不对报文做任何处理；MAG收到回传的时戳信息之后进行时戳恢复。

步骤3 PPE恢复时间戳信息。

MAG 收到回传的时间戳信息后，恢复出完整的 96bit 时间戳数据，格式为{3bit0, 5bit port_id, 8bit pid, 16bit seqinfo, 32bit回复后秒信息, 2bit原始秒信息, 30bit原始纳秒}。

步骤4 PPE 查询并通知 POE。

PPE 根据发送报文的 MAC 端口查表，得到{回传的POE location, 接收的vqi}。

PPE 以消息报文形式将时戳信息告知 POE。

步骤5 POE 通知 NP 并处理。

POE收到消息报文后通知NP。

NP读取该消息报文并进行相应处理。

----结束

4.4.4.4 时戳恢复算法

PPE会通过和RTC的本地时间做运算，将接收侧报文的随路时戳和发送侧回传的时戳进行恢复；恢复后的时间戳信息格式为：{8' b0,24bit Seq Info,32bit秒, 2bit原始秒, 30bit纳秒}；恢复算法为：

将MAC回传的秒时间记为S_s[1:0]，纳秒时间记为S_ns[29:0]；RTC送过来的秒时间记为R_s[31:0]，纳秒时间记为R_ns[31:0]。

- S_s[1:0]≤R_s[1:0]，则{timestamp_s,timestamp_ns}={{ R_s[31:2], S_s[1:0]}, { S_s[1:0], S_ns[29:0]}}。
- S_s[1:0]>R_s[1:0]，则{timestamp_s,timestamp_ns}={{ R_s[31:2]-1, S_s[1:0]}, { S_s[1:0], S_ns[29:0]}}。

4.4.4.5 LSW 和直通分发的 QoS 处理

4.4.4.5.1 入口 CoS 生成

由于各端口高低优先级定义或高低优先级个数不一致，所以入口报文需要映射到统一定义的CoS。

入口可以基于MAC_TNL/PPE_TNL选择使用外层或内层VLAN.PRI（如果入口流表处理过选择的那层VLAN，使用处理过的外层VLAN.PRI，如果没有选择的那层VLAN，使用该入口配置的PRI）查入口CoS表（26个PPE_TNL和26个MAC_TNL），生成内部统一优先级的CoS。



出口PRI生成

- 如果出口流表设置修改或增加入口CoS处理的那层VLAN的PRI，从流表的表项得到PRI。
- 不满足上一条的报文，使用报文的入口生成的CoS和报文的类型：单播、多播、广播，查对应出口PRI表（26个MAC_TNL*3种报文类型共78个表）。

4.4.4.5.2 出口 TC 生成

26个MAC出口，都支持最多4个TC的ETS调度。

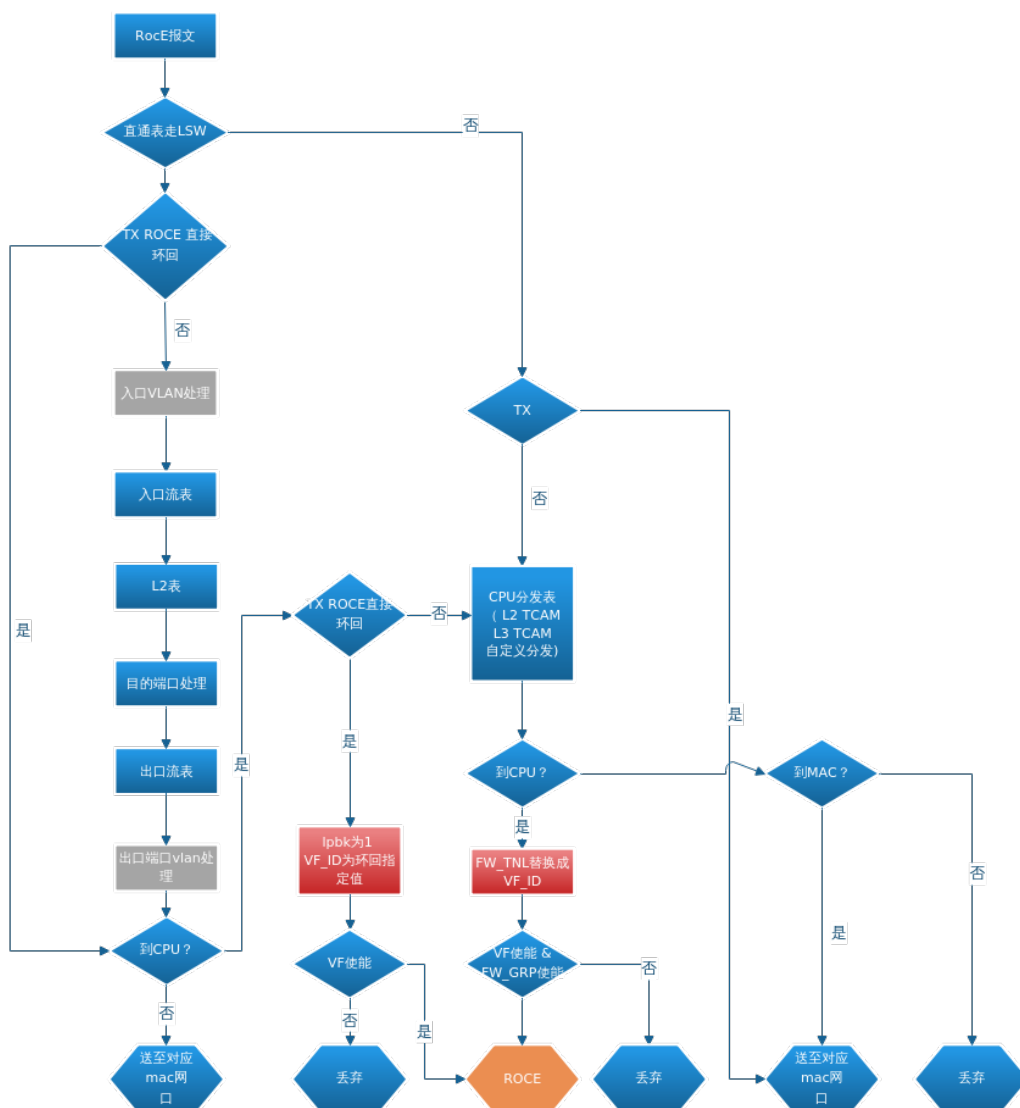
使用出口PRI查TC表生成TC值（26个MAC共26张表）。

须知

LSW结果是到PPE_TNL，查分发表后结果为到Mac的，使用入口得到Cos映射TC。

4.4.4.6 ROCE 报文处理

图 4-13 ROCE 在 PPE 转发处理流程



RoCE直接环回时的PPE_TNL号由TX的TNL得到，仅支持相同TNL的环回和在LSW环回，vfid为直接环回指定vfid， loopback标记会置1， 端口隔离不会隔离。

RoCE lsw普通环回通过配置LSW表项环回，TX查表的KEY由SSU和TPU配合提取（具体看），由mac和VLAN环到不同的tnl，vfid由查分发表得到，loopback不会置1。

LSW不对ROCE报文进行VLAN卸载功能，RX的VLAN检查由ROCE实现，检查加开关。

L5 id=8时是否识别为ROCE报文可以配置。

在发送方向时，查LSW表项的KEY由PPE模块进行提取，提取的格式主要如下：

对于RoCE报文TPU负责将KEY中的l5_id域填为8；KEY其他域TPU填写为0，由SSU负责提取。

由SSU负责给RoCE Tx报文提取简易的KEY用于查表，内容如下：

- SSU根据KEY中的l5_id值判断是不是RoCE报文（l5_id=8时为RoCE报文）
- KEY中的sc域在RoCE报文时复用RoCE接口的lpbk域。
- KEY中的sc_target域RoCE报文时复用RoCE接口的VF_ID域。
- SSU通过RoCE的info信号取VLAN num做为KEY的VLAN level域；（因VLAN可能有2个，RoCE接口的info信号中VLAN flag域增加为2bit更名为VLAN num）
- SSU将dmac、smac、qinq、VLAN域从报文中提取出并放入KEY的dmac域。
- KEY中的f_ind、pkt_id、cge_port_pri、mac_tnl_id域保持和普通NIC报文一样的处理规则。

除以上描述的域外，KEY的其他域SSU都填为0给PPP。

须知

对TX方向硬RoCE，只支持通过配表发送到固定的网口。

4.4.4.7 通用报文解析

接收方向PPE对报文进行逐层解析，支持的基本封装格式如图4-14所示。

图 4-14 封装格式 A



图 4-15 封装格式 B



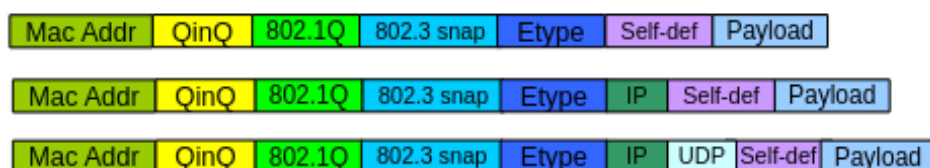
4.4.4.8 自定义报文解析

支持对EtherType、IP-Protocol、UDP/TCP-DortNum的值可配置（每个域支持15个配置值），识别出软件自定义的报文类型。

图 4-16 帧格式



图 4-17 帧格式





4.4.4.9 畸形报文的检查

- 以下情况上报二层错：
 - 报文的CRC检查错误。
 - 报文在二层头截断。
 - SNAP报文的长度域值大于后续的报文长度（不含CRC）；
 - VLAN中的CFI为1。
- 以下情况上报三层错：
 - IPv4、IPv6的版本号不正确。
 - 报文在IPv4、IPv6的头部截断；
 - IPv4的头长度IHL值小于5。
 - IPv4的Total Length小于4*IHL。
 - IPv4的非分片TCP报文的Total Length减去4*IHL小于20。
 - IPv4的非分片UDP报文的Total Length减去4*IHL小于8。
 - IPv4的Total Length大于IP头长度+实际IP的Payload长度。
 - IPv4的Header Checksum不正确。
 - IPv4的TTL域为0。
 - IPv4的非分片SCTP报文的Total Length减去4*IHL小于12。
 - IPv6的非分片TCP报文的Payload Length小于20。
 - IPv6的非分片UDP报文的Payload Length小于8。
 - IPv6的Payload Length大于实际IP的Payload长度。
 - IPv6的Hop Limit为0。
 - IPv6的非分片SCTP报文的Payload Length小于12。
- 以下情况上报四层错（只有非分片包才上报四层错）：
 - UDP的Length小于8。
 - UDP的实际Payload长度小于8。
 - UDP的Length大于实际UDP的Payload长度。
 - IPv4的UDP的Checksum不为0且不正确。
 - IPv6的UDP的Checksum为0或不正确。
 - TCP的实际Payload长度小于20。
 - TCP的Header Length大于IPv4的Total Length减去4*IHL或大于IPv6的Payload Length。
 - TCP的checksum不正确。

须知

- 当上报二层错误时，三层、四层错误指示为0。
- 当上报三层错误时，四层错误指示为0。
- 三层和四层错误，对于接收侧需要发送给ROCEE模块的报文会汇总为一个ERROE信号传递给ROCEE。



4.4.5 LSW 转发和 CPU 分发

4.4.5.1 直通表

在进去LSW和CPU分发查找之前，接口需要根据直通表的内容判断走LSW还是CPU分发查表。

表 4-4 接收直通表

接收方向直通表		
接收端口号	对应查表配置 1 bit	内部通道号5 bit（可配置）
Mac Port 0	直通标识，0是走LSW，1是走直通分发	PPE TNL 0
Mac Port 1	直通标识，0是走LSW，1是走直通分发	PPE TNL 1
Mac Port 2	直通标识，0是走LSW，1是走直通分发	PPE TNL 2
Mac Port 3	直通标识，0是走LSW，1是走直通分发	PPE TNL 3
Mac Port 4	直通标识，0是走LSW，1是走直通分发	PPE TNL 4
Mac Port 5	直通标识，0是走LSW，1是走直通分发	PPE TNL 5
Mac Port 6	直通标识，0是走LSW，1是走直通分发	PPE TNL 6
Mac Port 7	直通标识，0是走LSW，1是走直通分发	PPE TNL 7
Mac Port 8	直通标识，0是走LSW，1是走直通分发	PPE TNL 8
Mac Port 9	直通标识，0是走LSW，1是走直通分发	PPE TNL 9
...	...	...
Mac Port 24	直通标识，0是走LSW，1是走直通分发	PPE TNL 24
Mac Port 25	直通标识，0是走LSW，1是走直通分发	PPE TNL 25

另外发送发向也同样需要查发送直通表。



表 4-5 发送直通表

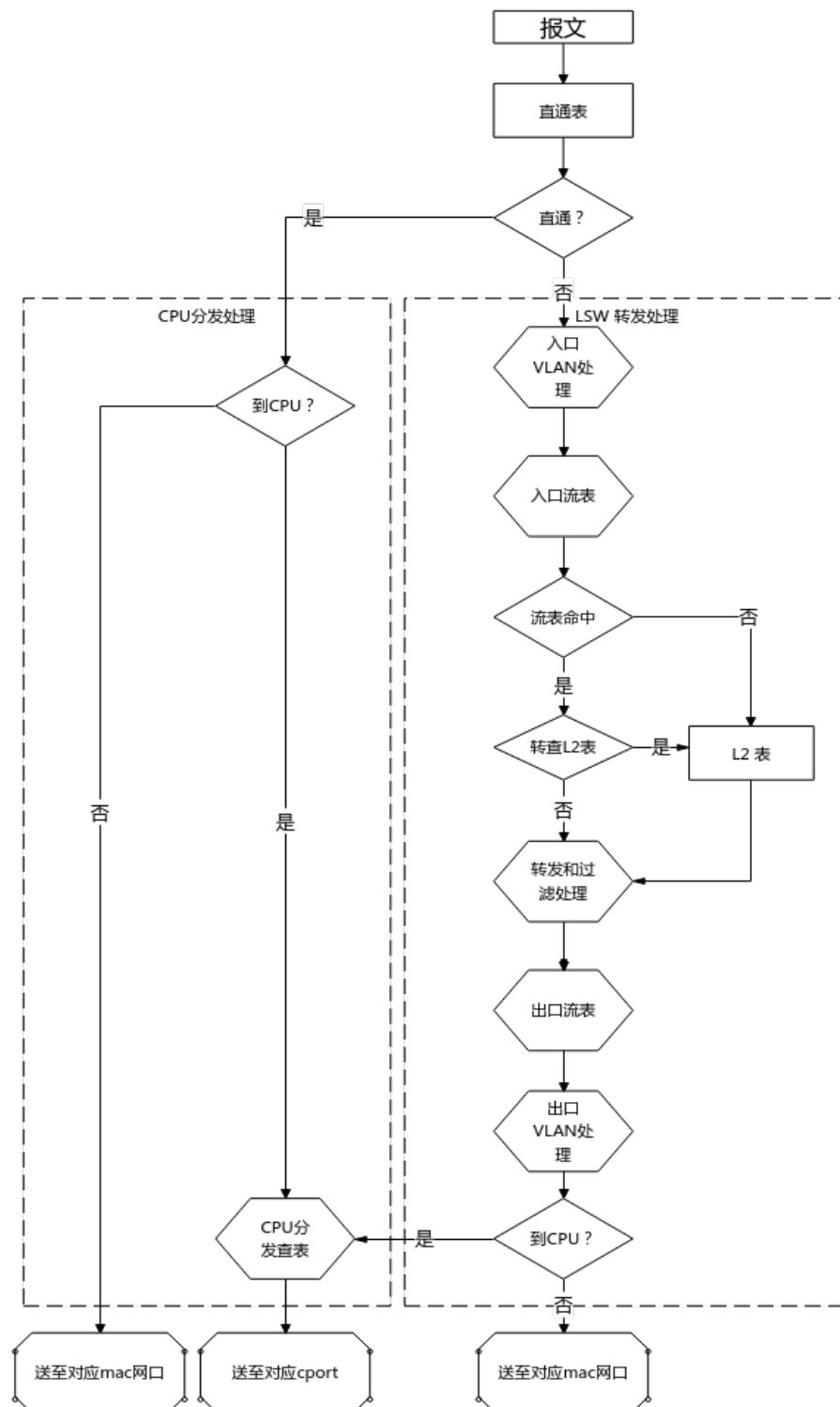
发送直通表		
发送通道号	对应查表配置 1 bit	外部MAC port号5 bit (可配置)
PPE TNL 0	直通标识, 0是走LSW, 1是走直通分发	MAC Port 0
PPE TNL 1	直通标识, 0是走LSW, 1是走直通分发	MAC Port 1
PPE TNL 2	直通标识, 0是走LSW, 1是走直通分发	MAC Port 2
PPE TNL 3	直通标识, 0是走LSW, 1是走直通分发	MAC Port 3
PPE TNL 4	直通标识, 0是走LSW, 1是走直通分发	MAC Port 4
PPE TNL 5	直通标识, 0是走LSW, 1是走直通分发	MAC Port 5
PPE TNL 6	直通标识, 0是走LSW, 1是走直通分发	MAC Port 6
PPE TNL 7	直通标识, 0是走LSW, 1是走直通分发	MAC Port 7
PPE TNL 8	直通标识, 0是走LSW, 1是走直通分发	MAC Port 8
PPE TNL 9	直通标识, 0是走LSW, 1是走直通分发	MAC Port 9
...	...	...
PPE TNL 24	直通标识, 0是走LSW, 1是走直通分发	MAC Port 24
PPE TNL 25	直通标识, 0是走LSW, 1是走直通分发	MAC Port 25

4.4.5.2 CPU 分发



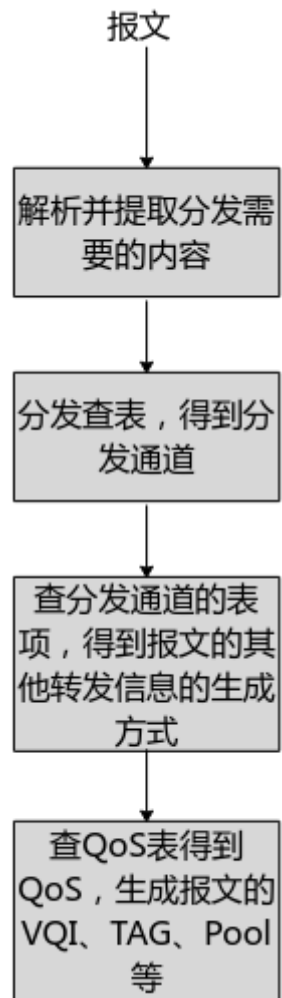
4.4.5.2.1 分发概述

图 4-18 报文转发流程处理



如上图4-18所示“CPU分发处理”流程中，从MAC进来的报文去往CPU，从CPU来的报文去往MAC。

图 4-19 报文分发流程表



4.4.5.2.2 分发规则

接收报文后，进行多级查表，将不符合规则的报文进行丢弃，符合规则的报文则生成VFID、VQI、TT，以及QoS、Tag的生成方式。

图 4-20 分发流程图

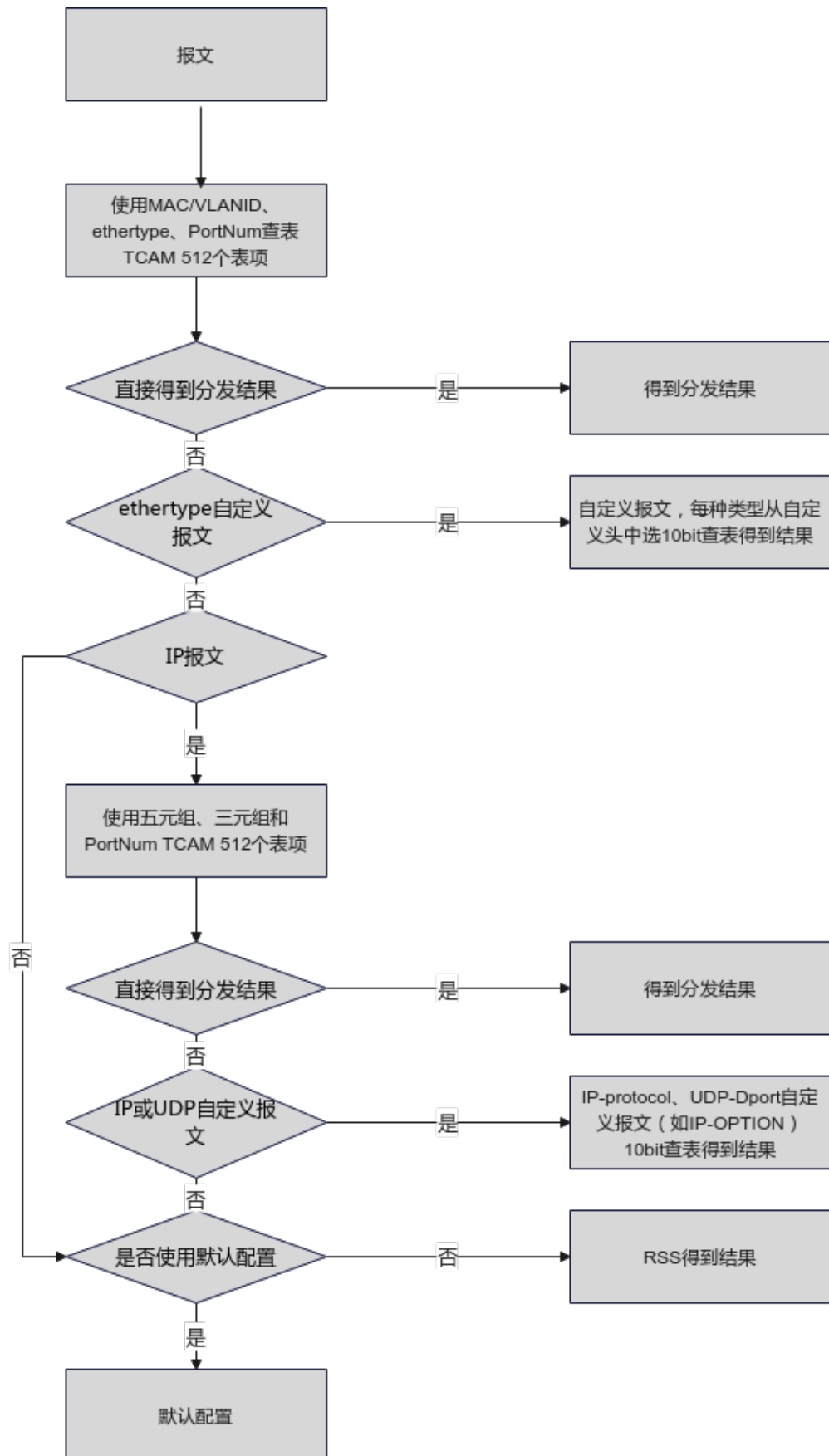




图4-20示中，第一级和第二级TCAM的KEY选择配置为推荐配置，非固定选择。

RSS请参见“RSS功能”章节。

分发表中基于报文元组来得到队列号的表项采用TCAM实现，这部分称为分发表。

分发表包括输入key和动作，在输入的key匹配流表中的表项后，该表项的动作将起作用。分发表支持两级结构，两级的key选择配置都是独立的。

因为TCAM的位宽是160bit，而输入的数据包括隧道外和隧道内的报文头远大于TCAM的位宽，因此需要从输入的数据选出160bit作为最后查表的key。

也可以配置为320bit。分发表模板匹配和key选择配置请参考入口流表处的说明。

key基于tuple粒度的产生，如下表4-6所示。

ot - 指隧道外层报文头信息。

in - 指隧道内层报文头信息。

表 4-6 分发表基于元组产生 key 的待选数据格式

Input keys (Note1)	Width (bit)	Description
ot_dmac	48 (MSB)	DMAC (网络).
ot_smac	48	SMAC.
ot_vlan_tag_outer	16	Outer VLAN TAG.
ot_vlan_tag_inner	16	Inner VLAN TAG.
ot_eth_type	16	Ether type.
ot_l2_udf	16	L2 reserve, parse by offset. The offset is calculate from the first Byte of the packet, and the extracted data can be any two-Byte data beyond the whole packet valid data, not limit to L2 Header.
ot_ip_tos	8	Type of servers.
ot_ip_proto	8	Protocol field for IPv4 packet and Next header field for IPv6 packet.
ot_sip	128	IPv4 or IPv6 source address. IPv6: All 128bits are valid. IPv4: bit[31:0]are valid, and bit 32~127 fix to 0.



Input keys (Note1)	Width (bit)	Description
ot_dip	128	IPv4 or IPv6 destination address. IPv6: All 128bits are valid. IPv4:bit[31:0]are valid, and bit 32~127 fix to 0.
ot_l3_udf	16	L3 Reserve, parse by offset. The offset is calculate from the first Byte of the packet, and the extracted data can be any two-Byte data beyond the whole packet valid data, not limit to L3 Header.
ot_sp	16	UDP source port.
ot_dp	16	UDP destination port.
ot_l4_udf	32	L4 Reserve, parse by offset. For a SCTP packet, the field is used to store the SCTP tag information. Or the offset is calculate from the first Byte of the packet, and the extracted data can be any four-Byte data beyond the whole packet valid data, not limit to L4 Header.
ot_tun_vni	24	VxLAN/GENEVE Network identifier or NVGRE Virtual Subnet ID.
ot_tun_flow_id	8	NVGRE Flow ID.
in_dmac	48	DMAC.
in_smac	48	SMAC.
in_vlan_tag_outer	16	Outer VLAN TAG.
in_vlan_tag_inner	16	Inner VLAN TAG.
in_eth_type	16	Ether type.



Input keys (Note1)	Width (bit)	Description
in_l2_udf	16	L2 reserve, parse by offset. The offset is calculate from the first Byte of the packet, and the extracted data can be any two-Byte data beyond the whole packet valid data, not limit to L2 Header.
in_ip_tos	8	Type of servers.
in_ip_proto	8	Protocol field for IPv4 packet and Next header field for IPv6 packet.
in_sip	128	IPv4 or IPv6 source address. IPv6: All 128bits are valid. IPv4: bit[31:0]are valid, and bit 32~127 fix to 0.
in_dip	128	IPv4 or IPv6 destination address. IPv6: All 128bits are valid. IPv4: bit[31:0]are valid, and bit 32~127 fix to 0.
in_l3_udf	16	L3 Reserve, parse by offset and length. (PA不再提取隧道内的l3_udf, 故不应使用该域参与key生成) The offset is calculate from the first Byte of the packet, and the extracted data can be any two-Byte data beyond the whole packet valid data, not limit to L3 Header.
in_sp	16	TCP/UDP/SCTP source port.
in_dp	16	TCP/UDP/SCTP destination port.



Input keys (Note1)	Width (bit)	Description
in_l4_udf	32	L4 Reserve, parse by offset. (PA不再提取隧道内的 l4_udf，故不应使用该域参与 key生成；) For a SCTP packet, the field is used to store the SCTP tag information. (SCTP报文时，此域可参与key的生成) The offset is calculate from the first Byte of the packet, and the extracted data can be any four-Byte data beyond the whole packet valid data, not limit to L4 Header.
dsa_tag	64	Dsa tag, Word0放在高位，Word1放在低位，4Byte时只有Word0.

在选择过程中，以8位（即1个Byte）为基本单位进行操作，若某Byte的选择标志被配置为1，则说明该Byte需纳入键值（Key）部分。考虑到TCAM的总位宽为160bit，这相当于20个Byte。有关详细信息，请参考入口流表处的说明。

表 4-7 分发流表的 Meta data

META_MASK bit	Input keys	Width (bit)	Description
13	Packet type ID	6	309a应用不建议选择。



META_MASK bit	Input keys	Width (bit)	Description
12	Packet type L5ID	4	0x0: DHCP/BOOTP报文; 0x1: IPPM报文; 0x2: UDP封装的1588报文; 0x3: GTPU报文; 0x4: HTTP报文; 0x5: HTTPs报文; 0x6: FTP报文; 0x7: BFD报文; 0x8: RoCE报文; 0x9~0xE: 保留; 0xF: 未知报文或基带自定义RAW报文。
11	Packet type L4 ID	4	0x0: UDP报文; 0x1: TCP报文; 0x2: GRE报文; 0x3: SCTP报文; 0x4: IGMP报文; 0x5: ICMP报文; 0x6: ESP报文; 0x7: AH报文; （IPSec包括AH和ESP） 0x8~0xE: 保留; 0xF: 未知报文或自定义RAW报文。



META_MASK bit	Input keys	Width (bit)	Description
10	Packet type L3 ID	4	0x0: IPv4报文; 0x1: IPv6报文; 0x2: ARP报文; 0x3: RARP报文; 0x4: 带option的IPv4报文; 0x5: 带扩展头的IPv6报文; 0x6: LLDP报文; 0x7: BPDU报文; 0x8: MAC PAUSE报文; 0x9: 保留; 0xA: MAC封装的1588报文; 0xB: 802.1x报文; 0xC: 保留 (1620用作CNM报文); 0xD: Y1731报文 (802.1ag, 两个eth_type一样使用同一个id); 0xE: OAM报文 (802.3ah); 0xF: 未知报文或自定义RAW报文。
9	ip fragement	1	是否为分片报文。 0: 非IP分片报文; 1: IP分片报文。 隧道报文时指隧道外层的是否是分片。



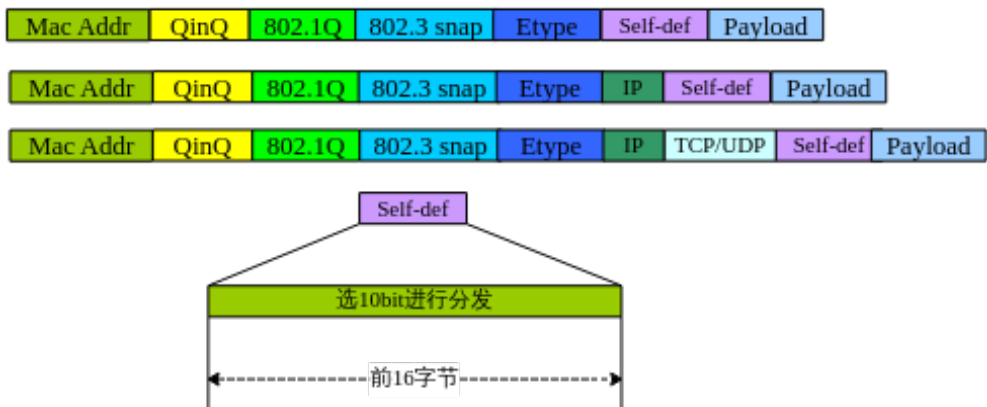
META_MASK bit	Input keys	Width (bit)	Description
8	nxt_key	5	bit[4:0]: 前一级的 next_key; 在第一级流表中需要 MASK 此域; 一级流表或者二级流表的第一级, 需要将此域 MASK 不使用; next_key 为 0 时表示不需要级联, 上一级的 next_key_vld 有效且 next_key 为非 0 表示需要级联。
7	ot_vlan_num	2	00: outer 无 tag, inner 无 tag; 01: outer 无 tag, inner 有 tag; 10: outer 有 tag, inner 无 tag; 11: outer 有 tag, inner 有 tag。
6	in_vlan_num	2	00: outer 无 tag, inner 无 tag; 01: outer 无 tag, inner 有 tag; 10: outer 有 tag, inner 无 tag; 11: outer 有 tag, inner 有 tag。
5	src_port	12	bit[11]: 报文来源 0 for CPU, 1 for MAC; 来源为 CPU 时: bit[10:9]: 为 0; bit[8:5]: reserved; bit[4:0]: ppe_tnl。
4	des_port	12	bit[9]: 报文目的 0 for CPU, 1 for MAC 目的为 CPU 时: bit[8:5]: reserved; bit[4:0]: ppe_tnl;



META_MASK bit	Input keys	Width (bit)	Description
3	Tunnel packet	1	定义是否为隧道报文。 0：非隧道报文； 1：隧道报文。
2	Mpls packet	1	是否是mpls报文，1有效，隧道报文时为隧道报文外层指示。
1	Template_id	5	key的模板id号。
0	crypt_dec	1	报文是否解密。 0：非解密； 1：解密。

自定义分发方式：

图 4-21 自定义分发生成示意图



- 在设置自定义报文的EtherType、protocol、UDP/TCP-Dport时，同时设置需不需要使用自定义分发模板，需要使用4个分发模板的哪个模板。
- 提取的分发的10bit只能在自定义头Self-def的前16Byte提取。
- 使用4个模板之一提取10bit方式。

如果同一个报文多层都使能了自定义提取功能，以最后一层为准。

IP分片报文（如IP-UDP分片），由于首片和非首片得到的自定义10bit可能不相同，为保证首片和非首片顺序进入POE，不应通过自定义分发的方式来进行分发；建议通过配置分发TCAM表实现首片和非首片进入相同的分发通道，保证首片和非首片进入同一个POE保序队列。



须知

1. 软件需要保证只能命中到一条分发结果，如果匹配到多个分发结果，使用第一个分发结果（低地址优先）进行分发。
2. 如果软件需要基于物理端口分发，将L2、L3表项内容中除src_port外的所有域都mask掉，实现该功能。
3. 对于隧道报文，如果选择使用隧道外层信息生成自定义信息，NVGRE报文自定义提取的位置不能超过IP头；其他隧道报文不能超过L4报文头。
4. TX_PA仅支持软件配置4个端口提取自定义L2、L3、L4UDF。
5. TX_PA仅支持软件配置4个端口提取10bit/6bit进行CPU分发/QOS生成。

表 4-8 分发流表的动作

分发表的相关数组		
域	位宽	描述
next_step	1	0：有效； 1：丢弃报文。
queue_id	9	next_step为0时，此域为队列号。
reserved	8	保留。
static_cnt_idx	5	流命中的统计cnt，在此表项命中时， 对应的cnt+1； 注意两级流表共享，建议不要两级流表 使用同一个cnt。
nxt_key	5	用下一级查表的key。
nxt_step_vld	1	是否进行下一级查表。 0b0：不查； 0b1：查下一级表。

第一级流表不丢弃且指示查下一级，那么会查下一级表，可能会命中ethtype自定义表，也可能命中第二级流表。如果想在第一级流表就得到分发结果，务必把nxt_step_vld配置为0。

如果需要级联，此时第二级流表需要选中meta data的nxt_key域。

表 4-9 自定义分发表的数据结构

分发表的相关数组		
域	位宽	描述
next_step	1	0：直接得到分发结果； 1：丢弃报文。

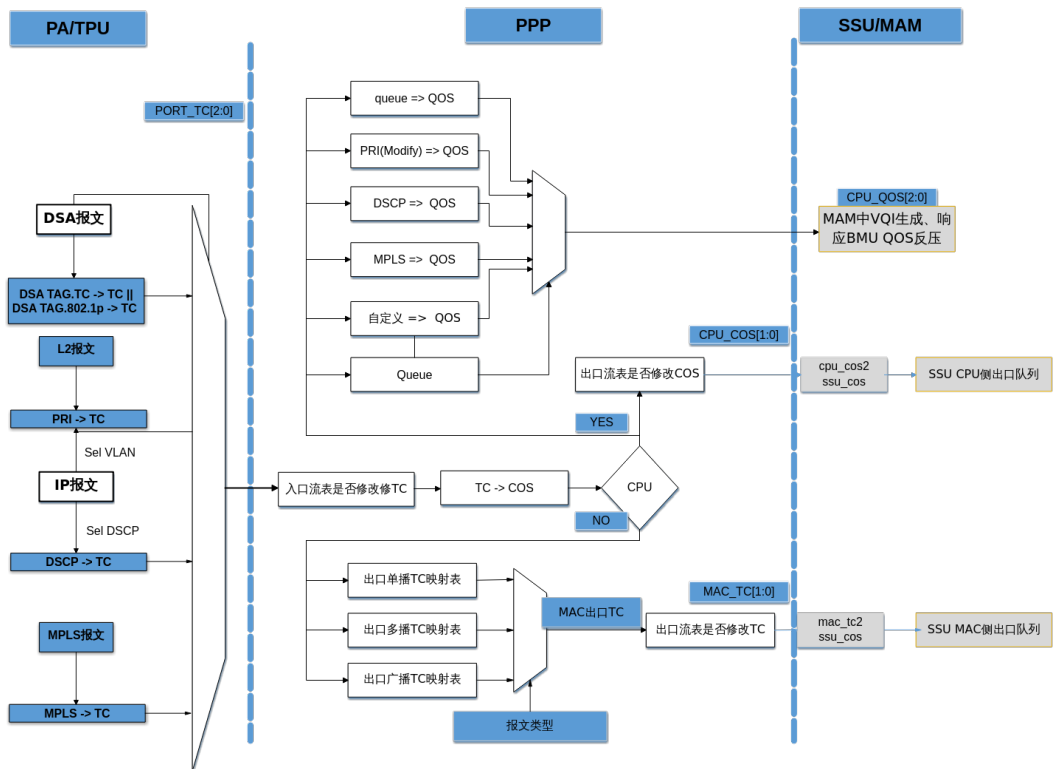


分发表的关联数组		
域	位宽	描述
queue_id	9	next_step为0时，为队列号。

4.4.5.2.3 QoS 生成

PPE需要对报文进行优先级映射和内部优先级管理；优先级映射管理的方法如下图4-23所示。

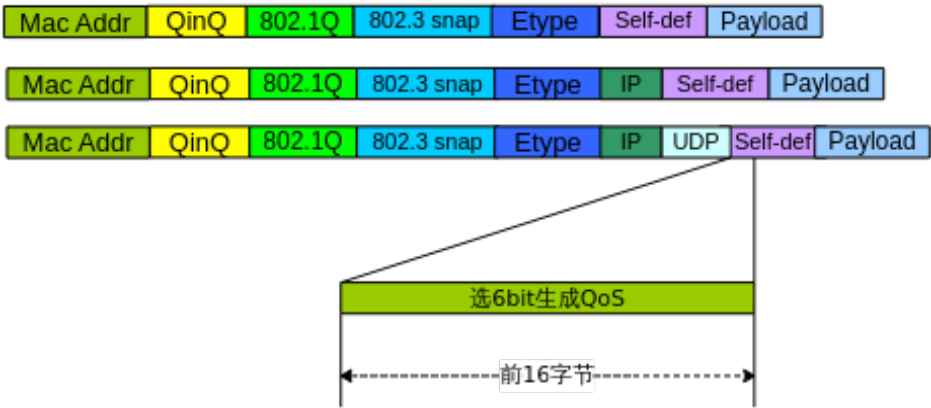
图 4-22 优先级映射管理



PPE按照分发结果选择QoS生成方式：

- MAC分发，选择PRI查表生成QoS或使用FW_TNL配置的QoS。
- IP三/五元组分发，软件可配置选择PRI或DSCP查表生成QoS或使用FW_TNL配置的QoS。
- 若报文为自定义类型分发，则采用自定义提取的Own_QoS查表生成QoS或使用FW_TNL配置的QoS。

图 4-23 自定义 QoS 生成示意图



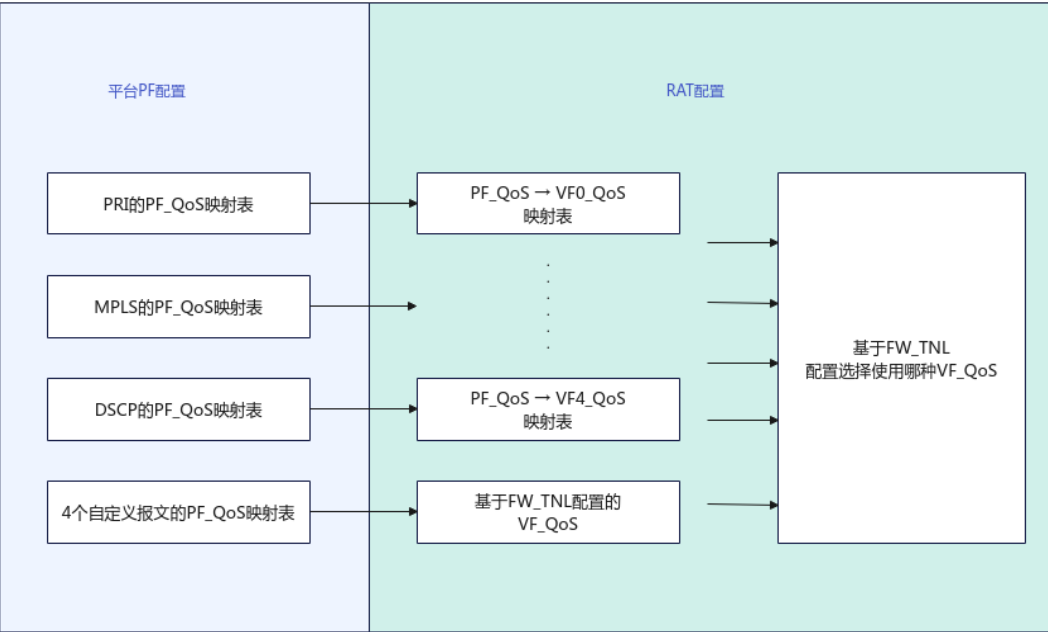
- 在设置自定义报文的EtherType、protocol、UDP/TCP-DortNum时，同时设置需不需要使用自定义提取QoS模板，需要使用4个提取QoS模板的哪个模板。
- 提取的分发的6bit只能在自定义头Self-def的前16Byte提取。
- 使用4个模板之一提取6bit方式查表生成。

须知

基于FW_TNL配置选择哪种QoS生成方式，需要软件保证QoS生成选择的正确性，例如：基于某FW_TNL配置的为基于MPLS_QoS生成QoS，则需要保证该FW_TNL对应的所有报文均为MPLS报文。

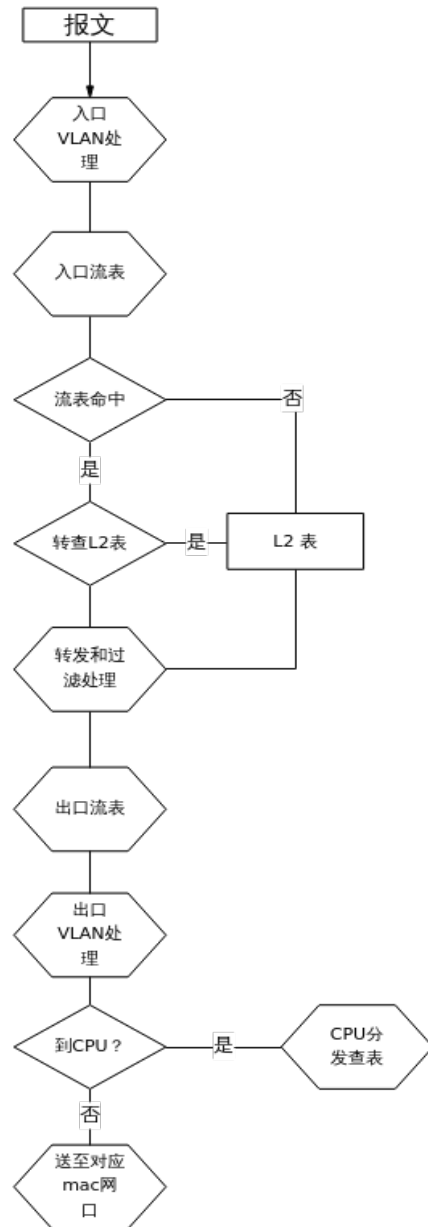
配置划分如下，PF负责全局的PF_QoS[2:0]表维护，每个VF有一个PF_QoS[2:0]映射到VF_QoS[2:0]的表。

图 4-24 QoS 的配置划分



4.4.5.3 LSW 转发

图 4-25 LSW 转发流程图



4.4.5.3.1 LSW VLAN 特性

PPE 支持对单层 VLAN 模式和 VLAN STACKING 的特性。在单 VLAN 模式下，每个端口支持配置为 Access、Trunk 和 Hybrid 三个端口模式，相应的 VLAN 处理如下表 4-10 所示。



表 4-10 单 VLAN 处理模式

端口类型	收发	Tag标签	处理方式
Access	收	有	丢弃（缺省）。
			可支持： 1. 端口PVID相等时转发，否则丢弃。 2. 或者不管是否相同均直接转发。
	发	无	标记上端口的PVID，转发。
		有	剥除帧的Vlan Tag后发送出去。
Trunk	收	有	判断端口是否允许该Vlan帧进入。允许则转发，否则丢弃。
		无	标记上端口的PVID，转发。
	发	有	若收帧的Vlan Tag和端口PVID相等则剥除Vlan Tag后发送，否则直接发送。
Hybrid	收	有	判断端口是否允许该Vlan帧进入。允许则转发，否则丢弃。
		无	标记上端口的PVID，转发。
	发	有	判断Vlan在端口是Untagged还是Tagged。若是Untagged，剥除帧的Vlan Tag后发送；若是Tagged，则直接发送。

在单VLAN模式下，可以选择外层或者内层VLAN作为处理对象。

默认情况下，添加的PVID也是需要检查的。

VID等于0时，可以选择是否替换成默认值。在不替换时，仅在查L2表和VLAN表（出口VLAN过滤）时使用默认值。

另外作为扩展，支持VLAN Stacking在LSW入口的添加、修改、替换和删除，并对可以选择对哪一层VLAN进行检查，如果该层VLAN在VLAN表中允许通过则转发，否则丢弃，在LSW出口进行添加、修改、替换和删除。

在入口处，VLAN的处理包括入端口处理和入口流表处理，端口处理对VLAN Stacking支持添加和替换。

表 4-11 Vlan Stacking 模式

PPP Ingress 端口VLAN处理						处理说明
添加配置		添加前 VLAN		添加后		
out	in	out	in	out	in	-



PPP Ingress 端口VLAN处理						处理说明
添加配置		添加前 VLAN		添加后		
0	0	0	0	-	-	用默认基于的VLAN配置查表。
0	1	0	0	-	-	内层添加。
1	0	0	0	-	-	外层添加。
1	1	0	0	-	-	内外层同时添加。
0	0	0	1	-	-	外层查表使用默认值，内层不变。
0	1	0	1	-	-	外层查表使用默认值，内层修改。
1	0	0	1	-	-	外层添加，内层不变。
1	1	0	1	-	-	外层添加，内层用端口配置值修改。
0	0	1	0	-	-	外层不变，内层查表使用默认值。
0	1	1	0	-	-	外层不变，内层添加。
1	0	1	0	-	-	外层用端口配置值修改，内层查表使用默认值。
1	1	1	0	-	-	外层用端口配置值修改，内层添加。
0	0	1	1	-	-	外层不变，内层不变。
0	1	1	1	-	-	外层不变，内层用端口配置值修改。
1	0	1	1	-	-	外层用端口配置值修改，内层不变。
1	1	1	1	-	-	外层用端口配置值修改，内层用端口配置值修改。

添加配置—0代表无添加，1代表添加；

添加前VLAN - 0代表原始报文不带VLAN，1代表带VLAN；

VLAN处理结果有4类：



- type=0b00 -- 不处理该层VLAN；
- type=0b01 – 添加VLAN；
- type=0b10 – 删除VLAN；
- type=0b11 – 修改VLAN；

入端口的结果有0b00， 0b01， 0b11种，无删除处理，每一层VLAN都有自己的处理结果。

入口流表支持的VLAN操作更加丰富，入口流表的VLAN处理优先级要高于入端口VLAN处理。具体请参考入口流表的关联数据的VLAN处理部分，这里就不详述了。

入口端口的VLAN和入口流表的处理需要合并，每一层的合并规则如下表4-12所示。

表 4-12 入口 VLAN 和入口流表处理合并

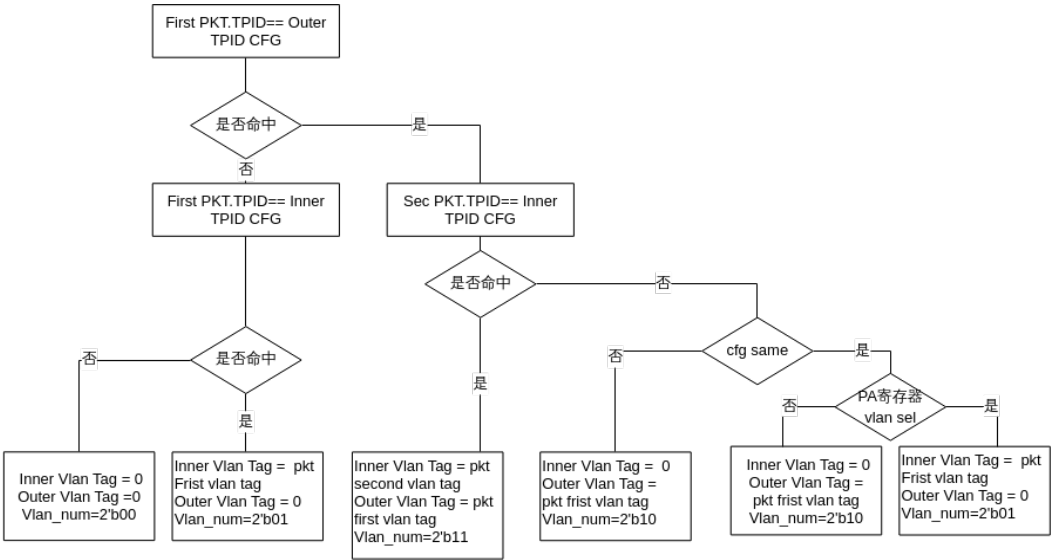
入口VLAN和入口流表处理合并					处理说明（以外层VLAN处理为例）
ingress_vlan		ingress_fd_vlan			type=00: 不处理 入口流表 vlan_modify=0b000; type=01: 添加 入口流表vlan_modify=0b010; type=10: 删除 入口流表vlan_modify=0b011; type=11: 修改 入口流表vlan_modify=0b001; 入口流表 vlan_modify=0b100/101/110 bypass=1: 入口流表vlan_modify=0b111。
type		bypass	type		-
0	0	0	0	0	不处理。
0	0	0	0	1	添加。
0	0	0	1	0	删除。
0	0	0	1	1	修改。
-	-	-	-	-	-
0	1	0	0	0	端口处理为添加，流表为不处理，结果为添加。
0	1	0	0	1	端口处理为添加，流表为添加，结果为添加，VLAN tag以流表的为准。



入口VLAN和入口流表处理合并					处理说明（以外层VLAN处理为例）
0	1	0	1	0	端口处理为添加，流表为删除，结果为不处理。
0	1	0	1	1	端口处理为添加，流表为修改，结果为添加，VLAN tag以流表的为准。
-	-	-	-	-	以下四个可以忽略，因为入口没有删除。
1	0	0	0	0	端口处理为删除，流表为不处理，结果为删除。
1	0	0	0	1	端口处理为删除，流表为添加，结果为修改，VLAN tag以流表的为准。
1	0	0	1	0	端口处理为删除，流表为删除，结果为删除。
1	0	0	1	1	端口处理为删除，流表为修改，结果为修改，VLAN tag以流表的为准。
-	-	-	-	-	-
1	1	0	0	0	端口处理为修改，流表为不处理，结果为修改。
1	1	0	0	1	端口处理为修改，流表为添加，结果为修改，VLAN tag以流表的为准。
1	1	0	1	0	端口处理为修改，流表为删除，结果为删除。
1	1	0	1	1	端口处理为修改，流表为修改，结果为修改，VLAN tag以流表的为准。
-	-	-	-	-	-
x	x	1	0	0	不管端口的处理，流表为不处理，结果为不处理。

在MAC接收侧和TPU发送侧，Vlan的识别如下图4-26所示。

图 4-26 VLAN 识别



TPID – 标签协议标识

VLAN tag – TCI

Vlan_num—VLAN存在指示，如果该层VLAN存在，值为1，否则为0。

在编辑VLAN时，只支持对最靠近最外层mac的两层VLAN进行编辑。

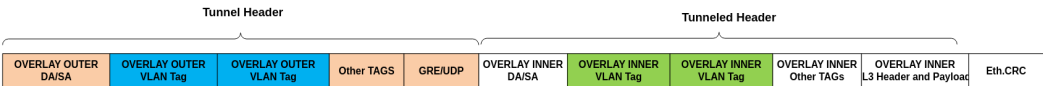
对于非隧道报文。

图 4-27 非隧道报文模式

DA/SA	Outer VLAN Tag	Inner VLAN Tag	Other TAGS	L3 Header and Payload	Eth.CRC
-------	----------------	----------------	------------	-----------------------	---------

对于隧道报文，处理的对象为隧道外层头的两层VLAN。

图 4-28 隧道报文模式



用户在单VLAN模式下可以选择内层还是外层作为处理对象，如果该层VLAN存在，值为1，否则为0。



表 4-13 单 VLAN 模式选择

外层可配置 (例如88A8)	内层可配置 (例如8100)	VLAN选择	access /trunk / hybrid 模式
0	0	内	入口没有VLAN
0	1	内	入口有VLAN，处理 内层
1	0	内	入口没有VLAN
1	1	内	入口有VLAN，处理 内层
0	0	外	入口没有VLAN
0	1	外	入口没有VLAN
1	0	外	入口有VLAN，处理 外层
1	1	外	入口有VLAN，处理 外层

在经过入端口和入口流表的VLAN处理后，有某层VLAN为0或者该层被删，所以定义了查后续表的值格式。0 表示该层无VLAN，1表示该层有VLAN。

表 4-14 默认值处理方式

入口处理后		-	-
外层	内层	查入口流表	-
0	0	内层=内层默认值，外 层=外层默认值	-
0	1	内层=处理值，外层=外 层默认值	-
1	0	内层=内层默认值，外 层=处理值	-
1	1	内层=处理值，外层=处 理值	-
入口流表处理后		查L2表	查出口流表
0	0	选择外层查表时用外层 默认值，选择内层时用 内层默认值	内层=内层默认值，外 层=外层默认值
0	1	选择外层查表时用外层 默认值，选择内层时用 处理值	内层=处理值，外层=外 层默认值



入口处理后		-	-
1	0	选择外层查表时用处理值，选择内层时用内层默认值	内层=内层默认值，外层=处理值
1	1	选择外层查表时用处理值，选择内层时用内层处理值	内层=处理值，外层=处理值

在出口处，端口对VLAN的处理主要是剥离，首先端口需要配置为剥离，然后选择哪一层去查表，那一层在剥离就需要看VLAN的tag_en配置，否则就仅看端口配置看是否剥离。

图 4-29 出口处 VLAN 处理

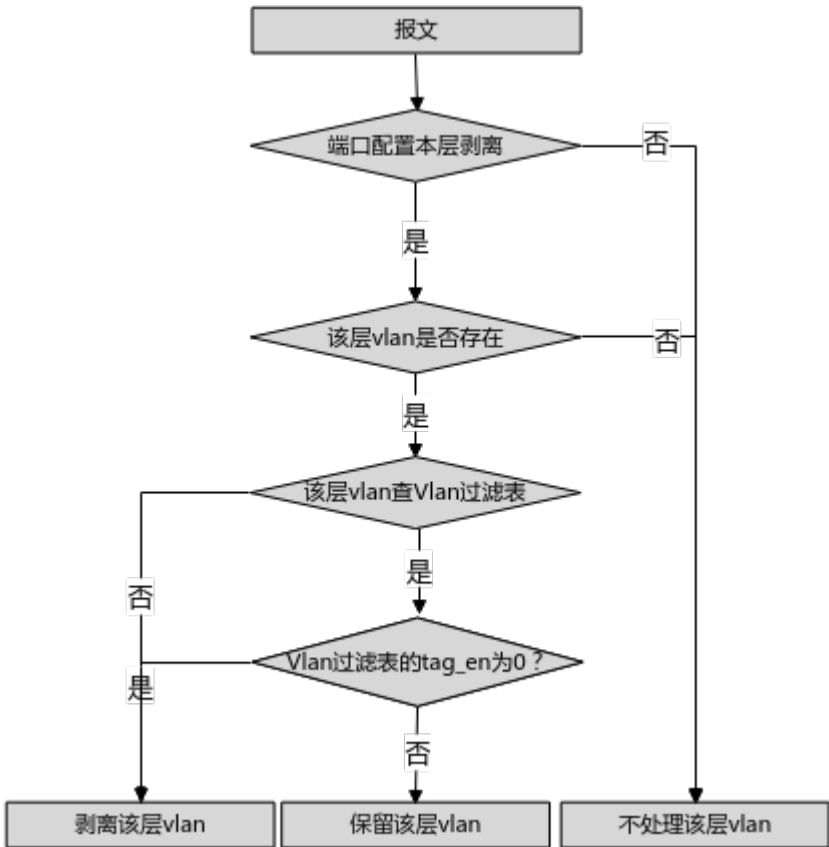




表 4-15 出端口 VLAN 处理

PPP Egress Port VLAN处理					
端口配置内层剥离	是否使用内层查表	Vlan表tag_en	剥离前是否有VLAN	剥离后	说明
0	0	0	0	-	不删除
0	0	1	0	-	不删除
0	1	0	0	-	不删除
0	1	1	0	-	不删除
0	0	0	1	-	不删除
0	0	1	1	-	不删除
0	1	0	1	-	不删除
0	1	1	1	-	不删除
1	0	0	0	-	不删除
1	0	1	0	-	不删除
1	1	0	0	-	不删除
1	1	1	0	-	不删除
1	0	0	1	-	删除
1	0	1	1	-	删除
1	1	0	1	-	删除
1	1	1	1	-	不删除
外层剥离配置	外层查表	Vlan表tag_en	剥离前	剥离后	-
0	0	0	0	-	不删除
0	0	1	0	-	不删除
0	1	0	0	-	不删除
0	1	1	0	-	不删除
0	0	0	1	-	不删除
0	0	1	1	-	不删除
0	1	0	1	-	不删除
0	1	1	1	-	不删除
1	0	0	0	-	不删除



PPP Egress Port VLAN处理					
端口配置内层剥离	是否使用内层查表	Vlan表tag_en	剥离前是否有VLAN	剥离后	说明
1	0	1	0	-	不删除
1	1	0	0	-	不删除
1	1	1	0	-	不删除
1	0	0	1	-	删除
1	0	1	1	-	删除
1	1	0	1	-	删除
1	1	1	1	-	不删除

在单VLAN模式时，只处理外层。

出口流表支持的VLAN操作更加丰富，出口流表的VLAN处理优先级要高于出端口VLAN处理，即出口有修改即以出口流表的为准。

具体请参考出口流表的关联数据的VLAN处理部分，这里就不详述了。

表 4-16 入口 VLAN 处理和出口 VLAN 处理合并规则

ingress_vlan		egress_vlan		处理说明
type		type		-
0	0	0	0	VLAN tag不处理
0	0	0	1	添加
0	0	1	0	删除
0	0	1	1	修改
-	-	-	-	-
0	1	0	0	入口处理为添加，出口为不处理，结果为添加
0	1	0	1	入口处理为添加，出口为添加，结果为添加，VLAN tag以出口的为准
0	1	1	0	入口处理为添加，出口为删除，结果为不处理，



ingress_vlan		egress_vlan		处理说明
0	1	1	1	入口处理为添加，出口为修改，结果为添加，VLAN tag以出口的为准
1	0	0	0	入口处理为删除，出口为不处理，结果为删除
1	0	0	1	入口处理为删除，出口为添加，结果为修改，VLAN tag以出口的为准
1	0	1	0	入口处理为删除，出口为删除，结果为删除，
1	0	1	1	入口处理为删除，出口为修改，结果为删除
1	1	0	0	入口处理为修改，出口为不处理，结果为修改
1	1	0	1	入口处理为修改，出口为添加，结果为修改，VLAN tag以出口的为准
1	1	1	0	入口处理为修改，出口为删除，结果为删除，
1	1	1	1	入口处理为修改，出口为修改，结果为修改，VLAN tag以出口的为准

在报文发到一个出口，带一层VLAN并需要替换外层tpid时，可以在入口流表和出口流表里用插入的方式配置，此种情况需要消耗较多流表表项。

4.4.5.3.2 流表

PPE的流表分为入口流表和出口流表，两张流表共享768x320bit的TCAM资源。TCAM分为24块tcam cell，每个cell规格为128x80bit；分配时需要考虑流表的位宽模式，例如：在入口流表为320bit时，那么需要分配4的倍数块给入口流表，为160bit时，需要分配2的倍数块给入口流表。



表 4-17 不同位宽流表对应最大深度

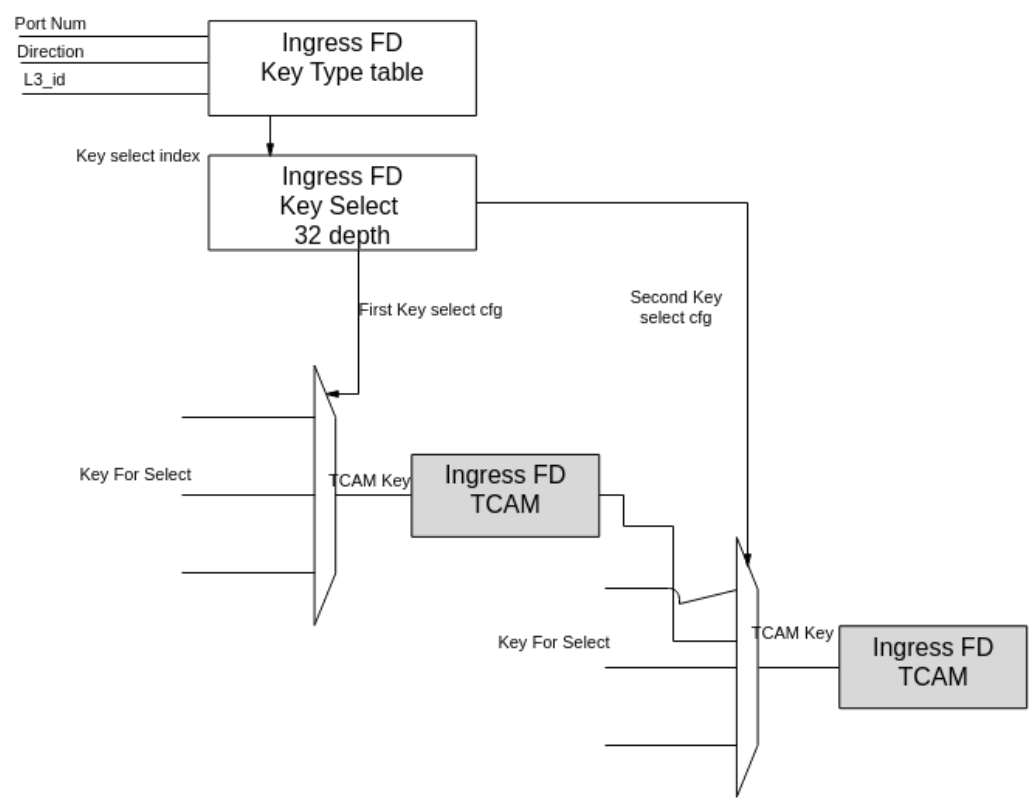
位宽模式	最大深度	最小深度
320bit	1024	0
160bit	2048	0
80bit	4096	0

因为有不同的bit位宽模式，所以分配的最小粒度为128x320bit。

4.4.5.3.2.1 流表的 key 选择

入口的key的待选数据如表4-18。考虑到报文类型多样性，以及key选择的配置粒度如果按接收方向的mac和发送方向通道分配，配置选择太多。因此，采用模板的共享的方式来实现，如下图4-30所示。

图 4-30 流表 key 选择结构图



所有Mac口和发送通道共享32深的模板匹配表，查表时，根据端口号、通道号、和l3_id得到模板的索引，再去把该模板的key选择配置读出，用来选择入口key待选数据。通过共享方式节省资源。每个模板支持32种key的选择配置。入口出口流表各有自己的选择配置，配置方式相同。



表 4-18 入口流表模板配置

深度	输入数据	输入数据	关联数据	关联数据
0	L3_ID 4bit 可mask	端口号、通道号 52bit bitmap	第一级模板号 5bit	第二级模板号 5bit
...	-	-	-	-
31	-	-	-	-

当无命中时，默认选择31表项的模板号。

入口流表支持两级，两级的模板的key选择是独立配置。

出口流表支持一级；

分发流表支持两级，两级的模板的key选择是独立配置。

4.4.5.3.2.2 入口流表

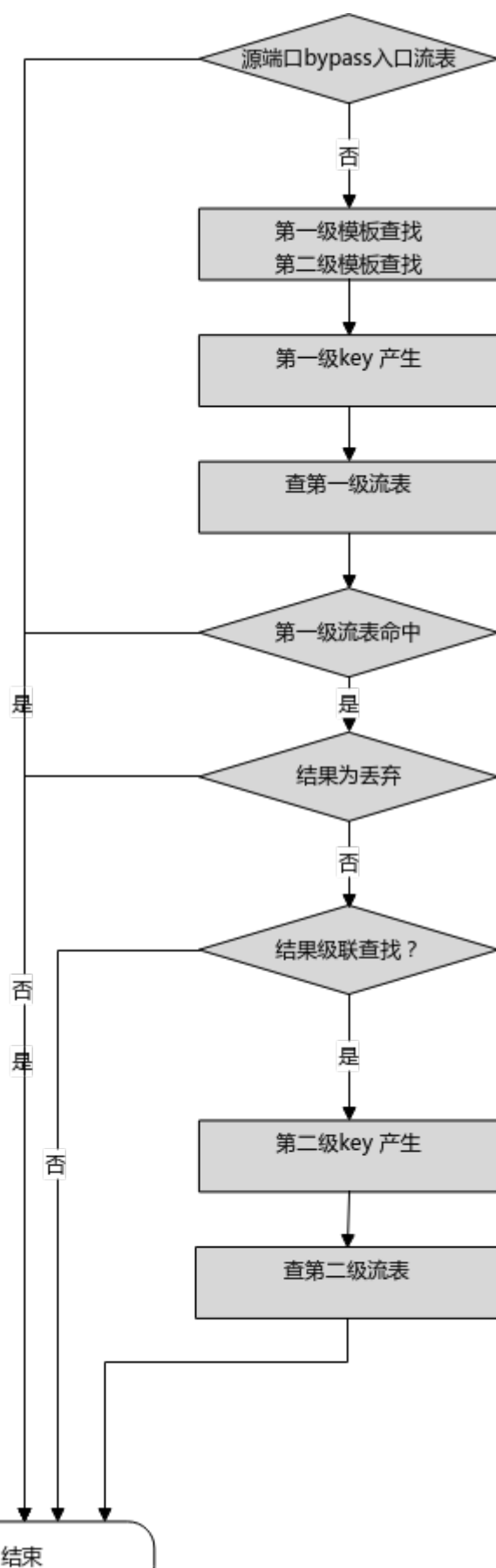
入口流表可以作为入口的ACL，可以实现转发、过滤、修改VLAN等动作。

流表的待选数据只能基于外层的信息做转发，即隧道报文时，基于隧道报文外层信息做转发。

入口流表可以级联查找，两级的key可以不一致，实现key位宽扩展。key一致时，可以深度扩展，需要在第一级最后一条表项指定查找下一级。查表流程如下图4-31所示。



图 4-31 查表流程





入口流表的key还可以用来对特殊报文的处理，例如捕获到某个CPU的ppe_tnl上。

因为最终的tcam的key位宽可配置为80/160/320bit模式，从待选数据中选好的meta和tuple域段一定要凑够tcam的位宽，不够硬件会补0，但是不能有交叠，因此配置时需要非常注意是否合理。

入口流表查表key数据如下表4-19所示。

表 4-19 入口流表 Key 待选数据

Meta data域	位宽 (bit)	描述
Pkt_crypt_dec	1 (41)	来自xsec。 0: 未经过解密; 1: 经过解密。
Template_id	5 (40)	key的模板id号
Pre_stage_hit	1 (35)	上一级流表是否命中。 0: 不命中; 1: 命中。
Flow_chain_id	8 (34)	上一级的flow_chain_id支持级联查找。
Ip_fragment	1 (26)	是否为分片报文。 0: 不是; 1: 是。
Tunnel_pkt	1 (25)	是否为隧道报文。 0: 非隧道报文; 1: 隧道报文。
Mpls_pkt	1 (24)	是否是mpls报文。 0: 非mpls报文; 1: mpls报文。
Dsa_pkt	1 (23)	是否是DSA报文。 0: 不是; 1: 是。



Meta data域	位宽 (bit)	描述
Ingress port	8 (22)	bit[7]IPv4或IPv6分片时，标识是否首分片。 0: 非首分片； 1: 首分片。 bit[6]: 类型说明。 0: for TX, from CPU; 1: for RX, from mac port。 bit[5:0]: 参考bit[6]。 TX: ppe_tnl 0~25; RX: mac 0~25。
Vlan_num	2 (14)	指示两层VLAN域是否有效，为入口VLAN处理后的值 00: outer无tag, inner无tag; 01: outer无tag, inner有tag; 10: outer有tag, inner无tag; 11: outer有tag, inner有tag。
L5_ID	4 (12)	0x0: DHCP/BOOTP报文; 0x1: IPPM报文; 0x2: UDP封装的1588报文; 0x3: GTPU报文; 0x4: HTTP报文; 0x5: HTTPs报文; 0x6: FTP报文; 0x7: BFD报文; 0x8: RoCE报文; 0x9~0xE: 保留; 0xF: 隧道报文。



Meta data域	位宽（bit）	描述
L4_ID	4（8）	0x0：UDP报文； 0x1：TCP报文； 0x2：GRE报文； 0x3：SCTP报文； 0x4：IGMP报文； 0x5：ICMP报文； 0x6：ESP报文； 0x7：AH报文；（IPSec包括AH和ESP） 0x8~0xE：保留； 0xF：未知报文或自定义RAW报文 隧道报文时： 0x0：非隧道报文； 0x1：MAC in UDP 隧道报文（VxLAN/ GENEVE）； 0x2：NVGRE隧道报文； 0x3：CAPWAP控制明文报文； 0x4：CAPWAP数据Keep-Alive明文报 文； 0x5：CAPWAP数据data payload明文报 文； 0x6：CAPWAP密文； 0x7~0xE：保留； 0xF：未知报文或自定义RAW报文。



Meta data域	位宽 (bit)	描述
L3_ID	4 (4)	0x0: IPv4报文; 0x1: IPv6报文; 0x2: ARP报文; 0x3: RARP报文; 0x4: 带option的IPv4报文; 0x5: 带扩展头的IPv6报文; 0x6: LLDP报文; 0x7: BPDU报文; 0x8: MAC PAUSE报文; 0x9: 保留 (1620用作PFC PAUSE报文); 0xA: MAC封装的1588报文; 0xB: 802.1x报文; 0xC: 保留 (1620用作CNM报文); 0xD: Y1731报文 (802.1ag, 两个 eth_type一样使用同一个id); 0xE: OAM报文 (802.3ah); 0xF: 未知报文或自定义RAW报文 隧道报文时: 0x0: IPv4报文; 0x1: IPv6报文; 0x2~0x3: 保留; 0x4: 带option的IPv4报文; 0x5: 带扩展头的IPv6报文; 0x6~0xE: 保留; 0xF: 未知报文或自定义RAW报文。
Tuple Data		
Dmac	48 (72)	Ethernet Destination address。
Smac	48 (66)	Ethernet Source address。
Outer Vlan Tag	16 (60)	outer VLAN tag, 无VLAN时使用pvid默认值。
Inner Vlan Tag	16 (58)	inner VLAN tag, 无VLAN时使用pvid默认值。
Eth_type	16 (56)	Ethernet type of the packet payload。
L2_UDF	16 (54)	L2 Reserve, parse by offset and length。
Ip_tos	8 (52)	Type of servers。



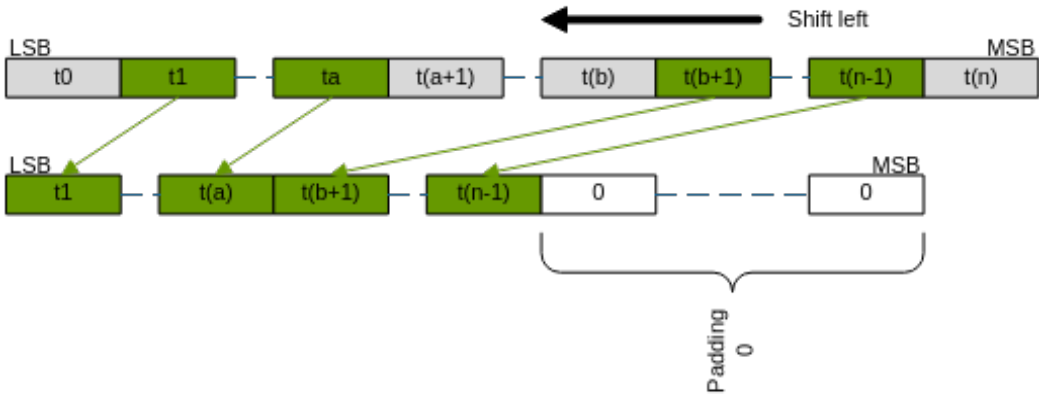
Meta data域	位宽 (bit)	描述
Ip_proto	8 (51)	IPv4 or IPv6 protocol number。
Ip_sp	128 (50)	IPv4 or IPv6 source address。
IP_dp	128 (34)	IPv4 or IPv6 destination address。
L3_UDF	16 (18)	L3 Reserve, parse by offset and length。
Sp	16 (16)	TCP/UDP source port。
Dp	16 (14)	TCP/UDP destination port。
L4_UDF	32 (12)	当in_l4_id == 3(sctp): sctp_tag 当 in_l4_id=0x6/0x7, spi 当in_l5_id=gtpu, teid else: L4 Reserve, parse by the offset which is configurable, extract a 4-Byte data。
DSA TAG	64 (8)	DSA TAG, Word0放在高位, Word1放 在低位, 4Byte时只有Word0。

在L3_ID=0或者L3_ID=1设置为当自定义报文而非IP报文处理时，IP相关的域段需要屏蔽。

在处理分片报文时，例如TCP报文，软件配置流表需要确保首片和非首片要转发到同一个目的地。

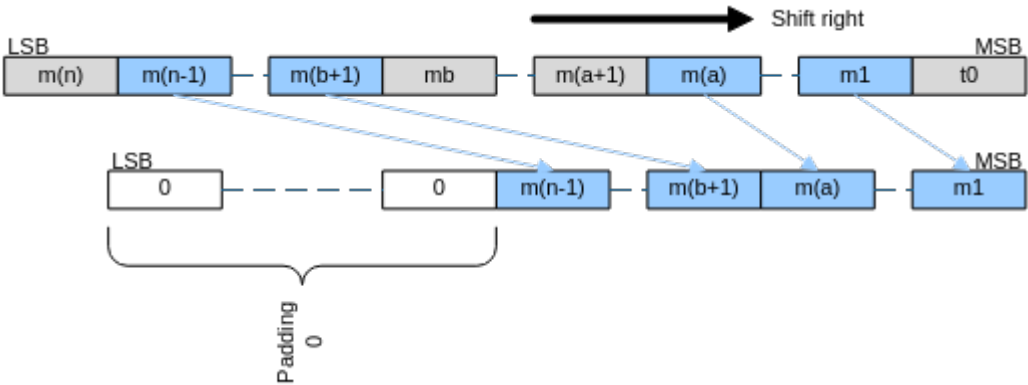
Dmac (MSB) ~ DSA_TAG (LSB) 这些为Tuple data域段，按照Byte为单位，每个单位一个使能配置，如果不使能则不作为TCAM查表的一部分，使能的字节将被左移对齐。

图 4-32 Tuple data 域段作为 TCAM 查表数据对齐方式



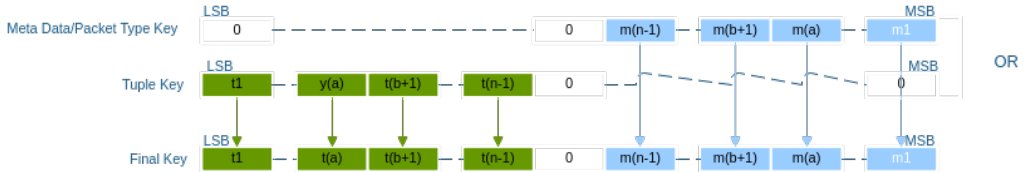
从L3_ID(LSB) ~ Pkt_crypt_dec (MSB) 这些为Meta Data域段，按照一个域段为单位，每个单位一个使能配置，如果不使能则不作为TCAM查表的一部分，使能的字节将被右移对齐。

图 4-33 Meta Data 域段作为 TCAM 查表方式对齐方式



移好的数据再进行相或，得到最终的key，软件需要保证两种key在选择时不能有重叠，否则会导致key产生错误。

图 4-34 Key 产生方式



入口流表的结构分两级查找，支持级联查找。入口流表匹配之后，报文会按照Action表项指示进行处理，Action表项如下表4-20所示。

表 4-20 入口流表关联数据

入口流表Action		
域	位宽 (bit)	描述
Next_step	2 (MSB) (96)	下一步处理方法： 00: 直接得到转发结果； 01: 保留； 10: 进入L2查表； 11: 丢弃报文。
Des_type	2(94)	Next_step为0时，报文的转发结果。 00: 单播非trunk； 01: 单播trunk； 10: CPU； 11: 多播。



入口流表Action		
域	位宽（bit）	描述
Des_id	11(92)	Des_type=0b00; Des_id[4:0]表示26个MAC端口; Des_id[5] 是否加密。 0: 不加密; 1: 加密。 Des_id[10:6]reserved; Des_type=0b01; Des_id[10:6] reserved; Des_id[5]是否加密。 0: 不加密; 1: 加密。 Des_id[4] 保留; Des_id[3] 保留; Des_id[2] 保留; Des_id[1:0]表示trunk ID, trunk 成员为0时会丢弃该报文。 Des_type=0b10; Des_id[10] reserved; Des_id[9] reserved; Des_id[8:5] reserved; Des_id[4:0] ppe_tnl; Des_type=0b11: Des_id[9:0]表示多播叶子MGID; Des_id[10]配置为0。



入口流表Action		
域	位宽 (bit)	描述
Inner_vlan_modify	3(81)	对报文的VLAN处理方法。 000: 不处理; 001: 替换VLAN-TAG, 使用Inner_vlan_tag替换报文的内层tag; 010: 插入VLAN_TAG, 使用Inner_vlan_tag插入报文中; 011: 删除VLAN_TAG, 删除报文的内层tag; 100: 只修改PRI,使用 Inner_Vlan_tag[15:13] 替换报文的内层tag的PRI; 101: 只修改VLAN ID, 使用 Inner_Vlan_tag[11:0]替换; 110: 只修改PRI和VLAN ID; 111: 不处理, 同时指示接口VLAN修改不处理。
Inner_vlan_tag	16(78)	内层编辑vlan_tag内容, 配合 Inner_vlan modify使用。
Outer_vlan_modify	3(62)	对报文的VLAN处理方法。 000: 不处理; 001: 替换VLAN-TAG, 使用 Outer_vlan_tag替换报文的外层tag; 010: 修改VLAN_TAG, 使用 Outer_vlan_tag插入报文中; 011: 删除VLAN_TAG, 删除报文的外层tag; 100: 只修改PRI,使用 Outer_Vlan_tag[15:13] 替换报文的外层tag的PRI; 101: 只修改VLAN ID, 使用 Outer_Vlan_tag[11:0]替换; 110: 只修改PRI和VLAN ID; 111: 不处理, 同时指示接口VLAN修改不处理。
Outer_vlan_tag	16(59)	外层编辑vlan_tag内容, 配合 Outer_vlan modify使用。



入口流表Action		
域	位宽 (bit)	描述
Mirror_en	1(43)	该报文需要mirror，与端口镜像是或的关系。 0x0：不需要mirror； 0x1：需要mirror。
Vlan_filter_en	1(42)	在出口处理时，是否需要进行VLAN 过滤。 0x0：不过滤； 0x1：过滤。
Ing_stp_filter_en	1(41)	在入口是否进行生成树过滤。 0x0：不过滤； 0x1：过滤。
Egr_stp_filter_en	1(40)	在出口是否进行生成树过滤。 0x0：不过滤； 0x1：过滤。
Loop_back_en	1(39)	是否进行允许源端口或源trunk环回，与端口环回配置是或的关系。 0x0：不允许； 0x1：允许。
Flow counter index	5(38)	流表共有32个计数器，表项命中时计数器加1，出口和入口流表共享。 注意：出口和入口流表不要使用相同的计数器，入口流表两级之间也不要使用相同计数器。
Smac_Dmac_swap	1(33)	现不支持。
Smac_modified_en	1(32)	现不支持。
Smac_modified_idx	5(31)	现不支持。
Dmac_modified_en	1(26)	现不支持。
Dmac_modified_idx	7(25)	现不支持。
Flow_chain_vld	1(18)	流表是否级联。 0：否； 1：是。
Flow_chain_id	8(17)	在级联时用于给第二级的key组成。



入口流表Action		
域	位宽（bit）	描述
TC_modified_en	1(9)	修改入口TC值。 0x0：不修改； 0x1：修改。
TC	3(8)	修改的值。
Rate_limit_car_id	5（LSB）	使用过滤的Car Id。

两级流表的处理规则：

- 第一级为丢弃时，不查第二级。
- 第一级和第二级都命中时，以第二级的结果为准。
- 第一级不命中时，不查第二级（在级联时需要在第一级配置一条表项，flow_chain_vld等于1）。
- 第一级命中，查第二级，第二级不命中，使用第一级的结果。

4.4.5.3.2.3 出口流表

出口流表的模板也是支持32种模板，32个模板配置，与入口流表配置类似。出口流表只支持一级。

表 4-21 出口流表 Key 待选数据

Meta Data		
Pkt_crypt_dec	1	来自xsec。 0：未经过解密； 1：经过解密。
Template_id	5	key的模板id号。
pkt_type	2	出口报文类型。 0x0：单结果； 0x1：多结果； 0x2：广播； 0x3：洪泛。
SA_Miss	1	在查L2表时是否命中SMAC。 0x0：命中； 0x1：未命中。



Meta Data		
egress_port	7	bit[6]: forward target。 0x0: to CPU; 0x1: to Mac; bit[6]=0: egress_port [4:0]表示ppe_tnl, 最大值为25; bit[6]=1: egress_port [4:0]表示26个目的MAC端口), 最大值为25。
ip_fragment	1	是否为分片报文。 00: 否; 01: 是。
tunnel_pkt	1	是否为隧道报文。 00: 非隧道报文; 01: 隧道报文。
Mpls_pkt	1	是否是mpls报文。 00: 非mpls报文; 01: mpls报文。
Dsa_pkt	1	是否是DSA报文。 0: 否; 1: 是。
Ingress port	8	bit[7]: IPv4或IPv6分片时, 标识是否首分片。 0: 非首分片; 1: 首分片。 bit[6] 0: for TX , from CPU; 1: for RX, from mac port。 bit[5:0] TX: ppe_tnl 25~0。 RX: mac 25 ~0。
vlan_num	2	指示两层VLAN域是否有效。 00: outer无tag, inner无tag; 01: outer无tag, inner有tag; 10: outer有tag, inner无tag; 11: outer有tag, inner有tag; 为入口VLAN处理后的值。
L5_ID	4	见入口流表相关域。
L4_ID	4	见入口流表相关域。



Meta Data		
L3_ID	4	见入口流表相关域。
Tuple Data		
Dmac	48	Ethernet Destination address。
Smac	48	Ethernet Source address。
Outer Vlan Tag	16	outer VLAN tag，无VLAN时使用pvid默认值。
Inner Vlan Tag	16	inner VLAN tag，无VLAN时使用pvid默认值。
Eth_type	16	Ethernet type of the packet payload。
L2_UDF	16	L2 Reserve，parse by offset and length。
Ip_tos	8	Type of servers。
Ip_proto	8	IPv4 or IPv6 protocol number。
Ip_sp	128	IPv4 or IPv6 source address。
IP_dp	128	IPv4 or IPv6 destination address。
L3_UDF	16	L3 Reserve，parse by offset and length。
Sp	16	TCP/UDP source port。
Dp	16	TCP/UDP destination port。
L4_UDF	32	当in_l4_id == 3(sctp): sctp_tag 当 in_l4_id=0x6/0x7, spi 当in_l5_id=gtpu, teid else: L4 Reserve，parse by the offset which is configurable，extract a 4-Byte data。
DSA TAG	64	DSA TAG，Word0放在高位，Word1放在低位，4Byte时只有Word0。

在配置key的选择时，需要注意：

- 对于tuple数据，Dmac为高位，DSA_TAG为低位，
- 对于meta数据，L3_ID为低位，Pkt_crypt_dec为高位。



表 4-22 出口流表关联数据

出口流表Action		
域	位宽（bit）	描述
Next_step	2 (MSB)	下一步处理方法。 00: 正常转发; 01: 重定向; 10: 预留; 11: 丢弃报文。
Des_type	1	Next_step为1时, 报文的重定向结果。 0x0: CPU; 0x1: 单播非trunk。
Des_id	11	Des_type=1b1; Des_id [4:0]表示26个MAC端口; Des_id[5] 是否加密; Des_id [10:6]保留; Des_type=1b0; Des_id[10:5] reserved; Des_id[4:0] ppe_tnl。
reserved	18	-
Inner_vlan_modify	3	对出口时报文的VLAN处理方法。 000: 不处理; 001: 替换VLAN-TAG, 使用Outer_vlan_tag替换报文的外层tag; 010: 插入VLAN_TAG, 使用Outer_vlan_tag插入报文中; 011: 删除VLAN_TAG, 删除报文的外层tag; 100: 只修改PRI, 使用Outer_Vlan_tag[15:13] 替换报文的外层tag的PRI; 101: 只修改VLAN ID, 使用Outer_Vlan_tag[11:0]替换; 110: 只修改PRI和VLAN ID; 111: 不处理, 同时指示出口VLAN修改不处理。
Inner_vlan_tag	16	内层编辑vlan_tag内容, 配合Inner_vlan_modify使用。



出口流表Action		
域	位宽（bit）	描述
Outer_vlan_modify	3	对出口时报文的VLAN处理方法。 000：不处理； 001：替换VLAN-TAG，使用Outer_vlan_tag替换报文的外层tag； 010：插入VLAN_TAG，使用Outer_vlan_tag插入报文中； 011：删除VLAN_TAG，删除报文的外层tag； 100：只修改PRI，使用Outer_Vlan_tag[15:13] 替换报文的外层tag的PRI； 101：只修改VLAN ID，使用Outer_Vlan_tag[11:0]替换； 110：只修改PRI和VLAN ID； 111：不处理，同时指示出口VLAN修改不处理。
Outer_vlan_tag	16	外层编辑vlan_tag内容，配合Outer_vlan modify使用。
Mirror_en	1	该报文需要mirror。 0x0：不需要mirror； 0x1：需要mirror。
TC_modified_en	1	修改出口TC值。 0x0：不修改； 0x1：修改。
TC	3	修改的值。
Flow counter index	5（LSB）	流表共有32个计数器，表项命中时计数器加1，出口和入口流表共享，注意出口和入口流表不要使用相同的计数器。

4.4.5.3.3 L2 表

4.4.5.3.3.1 L2 表概述

L2表指的是由MAC、VLAN_ID组成的转发表，完成二层的分发，同时完成DMAC地址过滤；VLANID 可以基于端口使用外层VLANID或内层VLANID查表。

L2表采用Hash的实现方式，支持3种Hash算法（简单XOR，微软toeplitz，CRC32），Hash的结果可以配置是否再次折叠。



L2表支持最大6K+128个表项，由1K个Hash桶和128个逃生桶组成。每个Hash地址的桶深为6，支持1K个hash地址，如果在一个hash地址hash冲突达到6次，则溢出到逃生桶。如果逃生桶满，则上报配置失败。

Mac和Vlan ID支持mask功能，Mac支持bit级屏蔽，Vlan ID可以按需要全部屏蔽。

另外L2表还支持按key或者直接地址方式读取关联数据和Key。

L2表的学习和漂移查找可以按端口来配置使能。使能打开时，进行学习查找，只对Smac为单播的报文进行学习，学习的值为{0b00，单播非trunk/单播trunk，des_id，0b1，0b0，0b1，0b0}。学习支持硬件学习和软件学习，软件学习支持最多缓存8个学习信息。如果学习查找命中，则看其是否发生了端口或者trunk漂移，发生则上报中断，如果是静态表项漂移，则丢弃报文，同样漂移也支持硬件回写或软件回写。

源MAC为多播、广播的报文和BPDU的报文不会被学习。

在硬件自学习时，还可以按端口或者trunk来限制学习的个数，避免学习风暴。

CPU配置时，会把学习查找的回写停止，避免多个源头更新L2表带来的数据一致性问题，但是中断也可以照常上报并统计因为CPU操作导致的学习和漂移不回写的个数。

另外过多的CPU操作也会影响L2表的性能。

L2表的Key如表4-23所示。

表 4-23 L2 地址表 Key

域	位宽	描述
PORT	5	报文的端口号，可按mac端口号隔离表项-，Hi309a屏蔽此字段。
VLAN_ID	12	报文的VLAN ID。
MAC	48	报文的MAC地址。

关联数据如下：

表 4-24 L2 地址表关联数据

域	位宽	描述
next_step	2 (MSB)	意义如下： 00：直接得到转发结果； 01：保留； 10：使用SMAC查找命中时丢弃；DMAC命中时为直接得到转发结果； 11：使用SMAC查找时为保留；使用DMAC查找时为丢弃（不支持DMAC或SMAC命中同时丢弃场景）。



域	位宽	描述
des_type	2	意义如下： next_step为0时，报文的转发结果； 0x0：单播非trunk，去网口； 0x1：单播trunk； 0x2：CPU； 0x3：多播。
des_id	11	Des_type=0b00； Des_id [4:0]表示26个MAC端口； Des_id[5] 是否加密，1加密，0不加密； Des_id [10:6]reserved； Des_type=0b01； Des_id[10:6] reserved； Des_id[5] 是否加密，1加密，0不加密； Des_id[4] 保留； Des_id[3] 保留； Des_id[2] 保留； Des_id[1： 0]表示trunk ID，trunk成员为0时会丢弃该报文； Des_type=0b10； Des_id[10:5] reserved； Des_id[4:0] ppe_tnl； Des_type=0b11； Des_id[9:0]表示多播叶子MGID； Des_id[10]配置为0。
limit_counted	1	自学习的表项是否计数，在被删除和替换时，对应的counter需要减1，只针对单播非trunk和单播trunk； 0x0： 非学习限制计数表项； 0x1： 学习限制计数表项。
static_item	1	静态表项指示，为1时，在MAC地址学习时不允许被替换掉。静态表项时建议老化使能配置为0。
aging_en	1	表项老化使能，1有效，在删除表项时，需要配置为0。



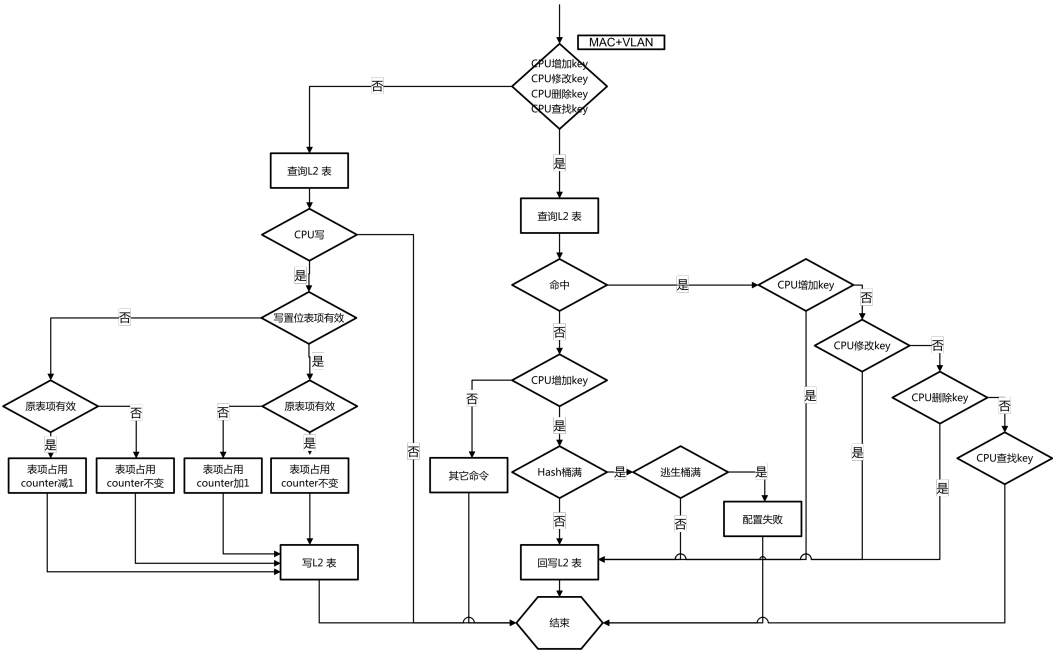
域	位宽	描述
aging_state	1 (LSB)	老化状态，1表示需要老化，支持由硬件或者软件老化，在硬件老化使能打开时，硬件维护此bit，如果由软件来老化，则把硬件老化使能关闭，由软件来置位。

next_step等于2时，在CPU配置表项时，相应的学习限制cnt不会加减。一般情况下不要和学习和漂移一起使用。

CPU 操作

CPU操作流程如下图4-35所示。

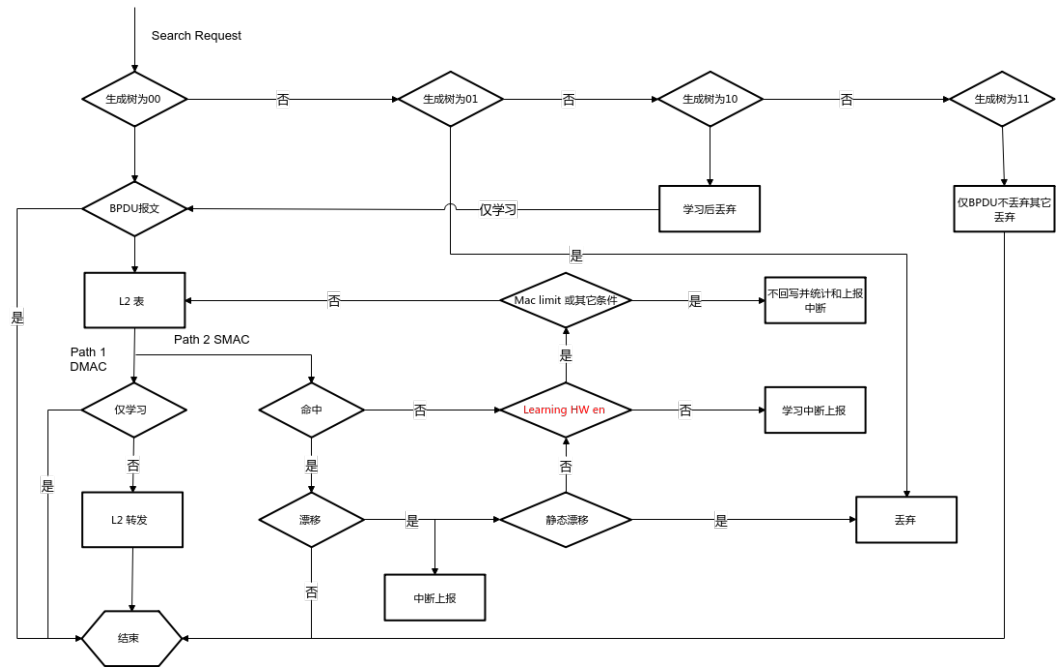
图 4-35 CPU 操作流程



学习和漂移

对于学习，基于MAC网口的配置学习查找使能。流程如图4-36所示。

图 4-36 L2 表学习和功能查找



当使能功能打开后，对于源MAC地址为单播地址的数据包，在L2表中会首先使用源MAC地址和VLAN ID进行查找，在此过程中，BPDU（Bridge Protocol Data Unit）报文的学习会被排除，以防止桥连接设备在连接时出现环路或漂移现象。若确实需要学习BPDU报文可以通过将报文发送至CPU，并由CPU通过添加特定的key来实现配置。此外，学习过程还会受到生成树协议状态的影响。

在L2表中完成源MAC地址查找后，如未命中（即没有找到匹配项），系统会进一步检查是否允许硬件学习，如果允许硬件学习且当前端口或者该端口所属的trunk组未超过学习限制，否则将该条目写入L2表；反之，若已达到学习限制，则不会写入新条目，同时进行相应的统计记录。

在非硬件学习时，硬件缓存最多可以同时处理或存储八个软件学习请求。

当接收到的数据包的源MAC地址命中现有MAC地址表项时，系统需要判断该MAC地址是否发生了漂移。漂移是指源MAC地址不再位于原先记录的端口或者trunk组内。如果确定发生了漂移，首先检查该表项是否是静态表项。如果是静态表项，则是发生了静态漂移，此时应丢弃该报文，若不是静态表项，则需要检查是否允许硬件回写。如果允许，则硬件会更新MAC地址表；如果不允许，则可能出发中断上报给软件处理。需要注意无论是否允许硬件回写，漂移事件都可以直接上报中断，这一点在流程图中未详细体现。

漂移的定义如下表4-25所示。

表 4-25 漂移表

原始来源	当前来源
port	trunk
trunk	port
trunk	trunk



原始来源	当前来源
port	port
cpu	trunk
cpu	port
port	cpu
trunk	cpu
cpu	cpu

目前系统支持MAC地址漂移回写功能。

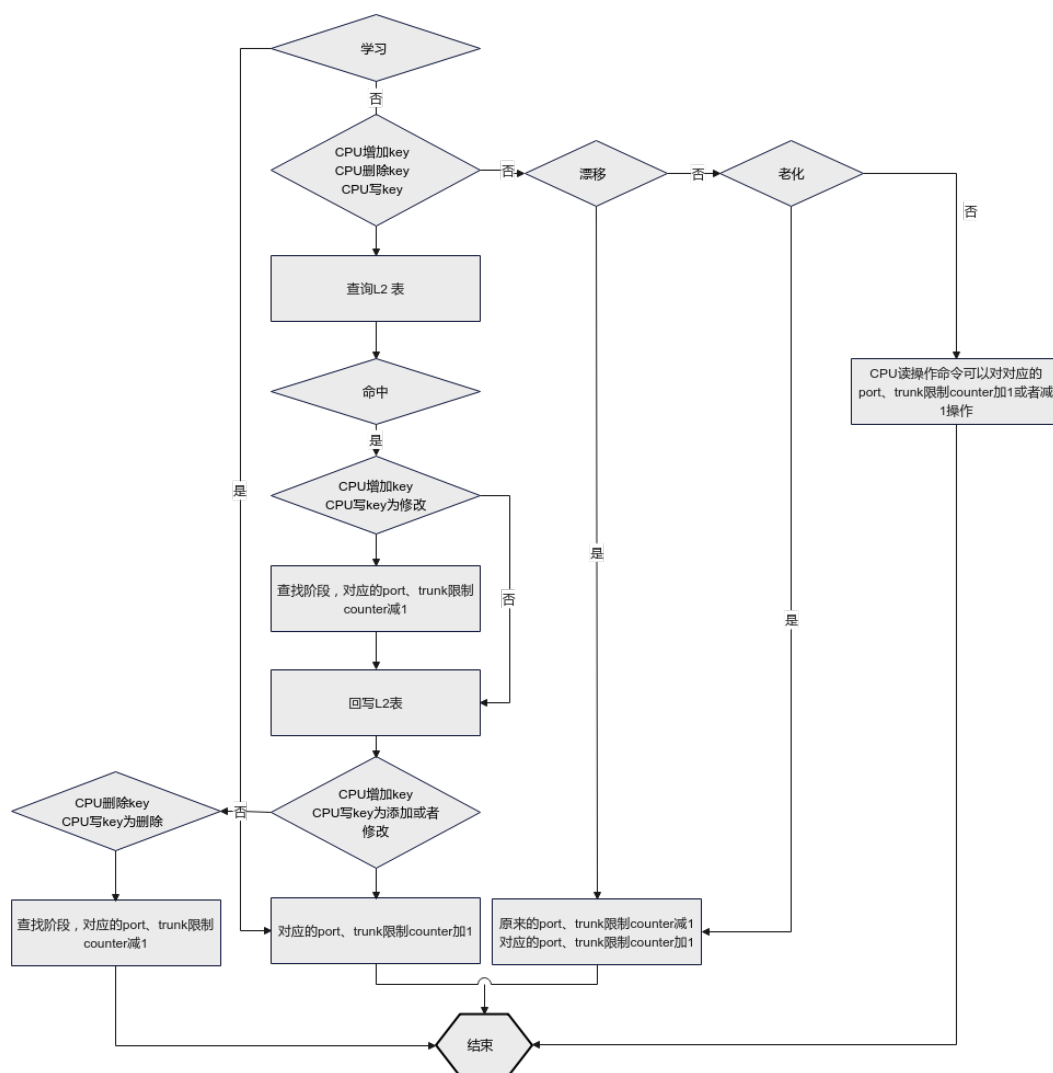
当表项从端口（Port）动态迁移至主干组（trunk Group）时，需要先删除原有表项，更新成员组信息后重新添加表项。

对于发生漂移的静态表项，应丢弃相关数据包，并向系统报告终端事件。

在Path1路径上，若接收到的数据包仅用于原地址的学习，则可直接跳过目的MAC地址发查找步骤，完成第二层（L2）表项查询，反之，若非仅学习类报文，则需利用目的MAC地址及VLAN ID进行匹配查找。

在MAC地址学习过程中，如通过硬件方式学习到新地址，对应端口（port）或者主干组（trunk）的限制计数器将递增1，并在地址解析数据库（AD）中设置相应位，以表明该表项和限制计数器相关联，在删除表项时，需要确保计数器值相应减少。

图 4-37 学习限制 counter 流程



当硬件学习成功时，相应的端口（port）或者干道（trunk）限制计数器加1。

4.4.5.3.4 老化

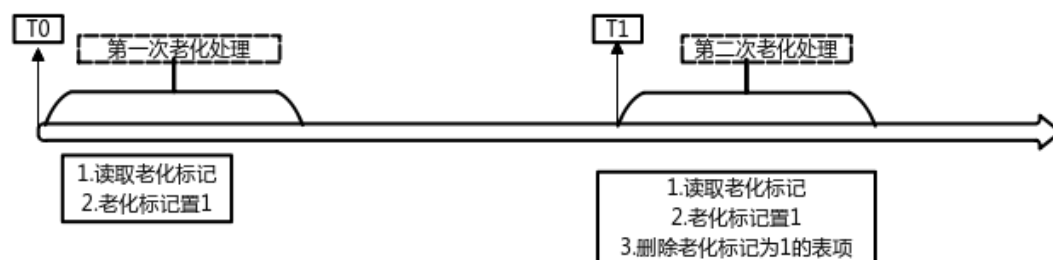
另外L2表还需要支持老化操作，老化标记和老化使能用另外的寄存器保存，不在L2的关联memory里，采用时间轮的老化机制，即一个timer和多个标记位。

初始时，每个老化标记被清0，表示不需要老化，当timer的时间间隔一到，把所有老化使能打开的老化标记都置位为1，如果下次timer的时间间隔一到，发现标记位还为1的，表示在很长时间内，这个表项没有被访问过，需要老化处理，将其删除。

老化的最大误差约为2个老化周期。

老化支持源MAC命中更新和目的MAC命中更新，可配置。

图 4-38 老化处理流程图



老化标记需要支持在CPU查找时设定不更新（某些场景下，CPU只是来查询一下，不希望影响老化状态）。

老化同样支持软件老化，由软件置位老化状态和维护老化时间，硬件更新老化状态，软件来删除需要老化的表项。

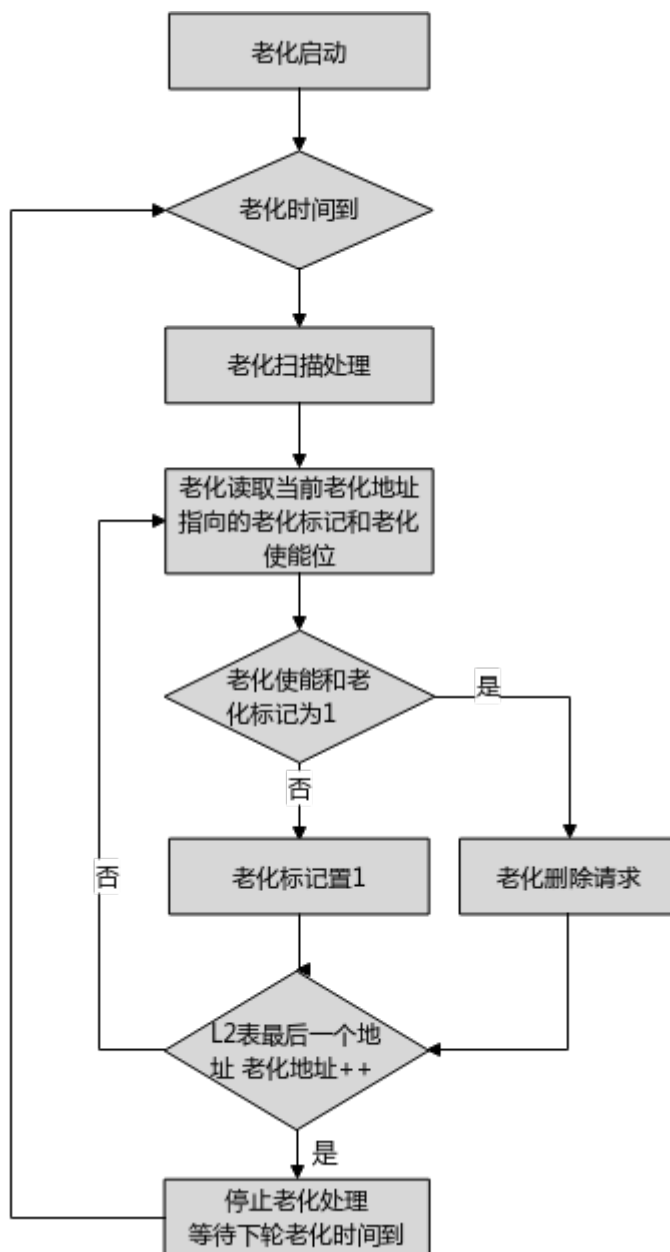
须知

老化的使能总开关在从0变化到1时，就会启动一次老化处理，如果当前配置的老化周期没到，把老化使能总开关配置为0，再配置为1，那么硬件检测到上升沿，就会启动老化处理，那么第一次老化处理时表项中老化状态标记为1的将被老化删除，从宏观上看，老化删除过早。如果只是改配老化时间，不改配老化使能，无论周期变长还是变短都是没有问题的。

因此建议：如果有老化开关变化，可以老化开关变成0后，软件把所有的表项的老化状态刷为0，或者等当前老化周期过了之后再改配。

老化开关不变化，只改配老化时间。

图 4-39 老化处理流程



4.4.5.3.5 生成树

生成树支持16个基于VLAN的生成树实例。每个端口包含两个bit的配置信息，以支持生成树的多个状态。

端口的生成树配置实现STP和RSTP的几种状态，如表[表4-26](#)、[表4-27](#)所示。



表 4-26 STP 状态表

端口状态	STP 描述	端口配置
Disabled 端口没有启用	此状态下端口不转发数据帧，不学习MAC地址表，不参与生成树计算。	0b01
Listening 侦听状态	此状态下端口不转发数据帧，不学习MAC地址表，只参与生成树计算，接收并发送BPDU。	0b11
Blocking 阻塞状态	此状态下端口不转发数据帧，不学习MAC地址表，此状态下端口接收并处理BPDU，但是不向外发送BPDU。	0b11，软件不发送
Learning 学习状态	此状态下端口不转发数据帧，但是学习MAC地址表，参与计算生成树，接收并发送BPDU。	0b10
Forwarding 转发状态	此状态下端口正常转发数据帧，学习MAC地址表，参与计算生成树，接收并发送BPDU。	0b00

表 4-27 RSTP 状态表

端口状态	RSTP 描述	端口配置
Discarding 丢弃状态	此状态下端口对接收到的数据做丢弃处理，端口不转发数据帧，不学习MAC地址表。 Alternate Port和Backup Port	0b01
Learning 学习状态	此状态下端口不转发数据帧，但是学习MAC地址表，参与计算生成树，接收并发送BPDU。	0b10
Forwarding 转发状态	此状态下端口正常转发数据帧，学习MAC地址表，参与计算生成树，接收并发送BPDU。	0'b00

在端口进入非转发状态时，即使该端口作为出站方向，也需要执行生成树过滤操作，入口流表指示特定管理报文是否需要生成树过滤，并且入口和出口的生成树过滤则需要单独设置。

MSTP中的每个端口状态是通过RSTP机制来实现。

4.4.5.3.6 Trunk（链路聚合）

系统支持单播Trunk选路功能，并通过软件配置多播表或硬件hash算法来实现多播Trunk的选路，默认采用硬件hash选路。

系统最多支持4个Trunk group，每个Trunk group最多支持8个成员，若某个Trunk组成员数量为0但流表和L2表项已配置了单播Trunk，则该组内的报文将会被丢弃，每个Trunk表内容如表4-28所示。



表 4-28 每个 Trunk group 内容

域	位宽 (bit)	描述
Trunk_member_number	4	该group的成员数。最小为1，最大为8。为0时表示没有成员。
Trunk_member_port_id0~3	5*8	每个成员的端口号。0~25去26个MAC。 ²

说明

2: CPU不会作为Trunk组成员。

- Hash选路支持两种方法：
 - 简单异或
 - CRC32
- 选路具体步骤如下：
 - 使用选定的hash算法计算出hash[5:0]；
 - 根据查表结果的trunkid选取对应的Trunk group内容；
 - 使用hash[5:0]% Trunk_member_number得到该报文去该trunk group的第几个成员；
 - 最后通过成员表查找并确定报文的具体出口。
- 简单异或算法

表 4-29 HASH 选路功能表

非IP报文	-	Hash=Mac_SA[5:0]^ Mac_DA[5:0]
IP报文	IP地址不参与计算	Hash=Mac_SA[5:0]^ Mac_DA[5:0]
IPv4	IP地址参与计算，非UDP和TCP，mac地址不参与计算	Hash==SIP[5:0]^SIP[21:16]^DIP[5:0]^DIP[21:16]
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit不参与计算，mac地址不参与计算	Hash=SIP[5:0]^SIP[21:16]^DIP[5:0]^DIP[21:16] ^l4_srcport[5:0]^l4_trgport[5:0];
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit参与计算，mac地址不参与计算	Hash=SIP[5:0]^SIP[21:16]^DIP[5:0]^DIP[21:16]^l4_srcport[5:0] ^l4_trgport[5:0]^l4_srcport[13:8]^l4_trgport[13:8];



非IP报文	-	Hash=Mac_SA[5:0]^ Mac_DA[5:0]
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit参与计算，mac地址参与计算	Hash=SIP[5:0]^SIP[21:16]^DIP[5:0]^DIP[21:16] ^l4_srcport[5:0]^l4_trgport[5:0] ^l4_srcport[13:8]^l4_trgport[13:8]^Mac_SA[5:0]^ Mac_DA[5:0];
IPv6	IP地址参与计算，非UDP和TCP，mac地址不参与计算	Switch(IPv6 Hash mode) Case0:IPV6_Hash=SIP[5:0]^SIP[21:16]^DIP[5:0]^DIP[21:16]; Case1:IPV6_Hash=SIP[125:120]^SIP[69:64]^DIP[125:120]^DIP[69:64]; Case2:IPV6_Hash = case_0^case_1;
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit不参与计算，mac地址不参与计算	Hash=IPV6_Hash^l4_srcport[5:0]^l4_trgport[5:0];
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit参与计算，mac地址不参与计算	Hash=IPV6_Hash^l4_srcport[5:0]^l4_trgport[5:0] ^l4_srcport[13:8]^l4_trgport[13:8];
-	IP地址参与计算，UDP和TCP参与计算，UDP和TCP高bit参与计算，mac地址参与计算	Hash=IPV6_Hash^l4_srcport[5:0]^l4_trgport[5:0] ^l4_srcport[13:8]^l4_trgport[13:8]^Mac_SA[5:0]^ Mac_DA[5:0];

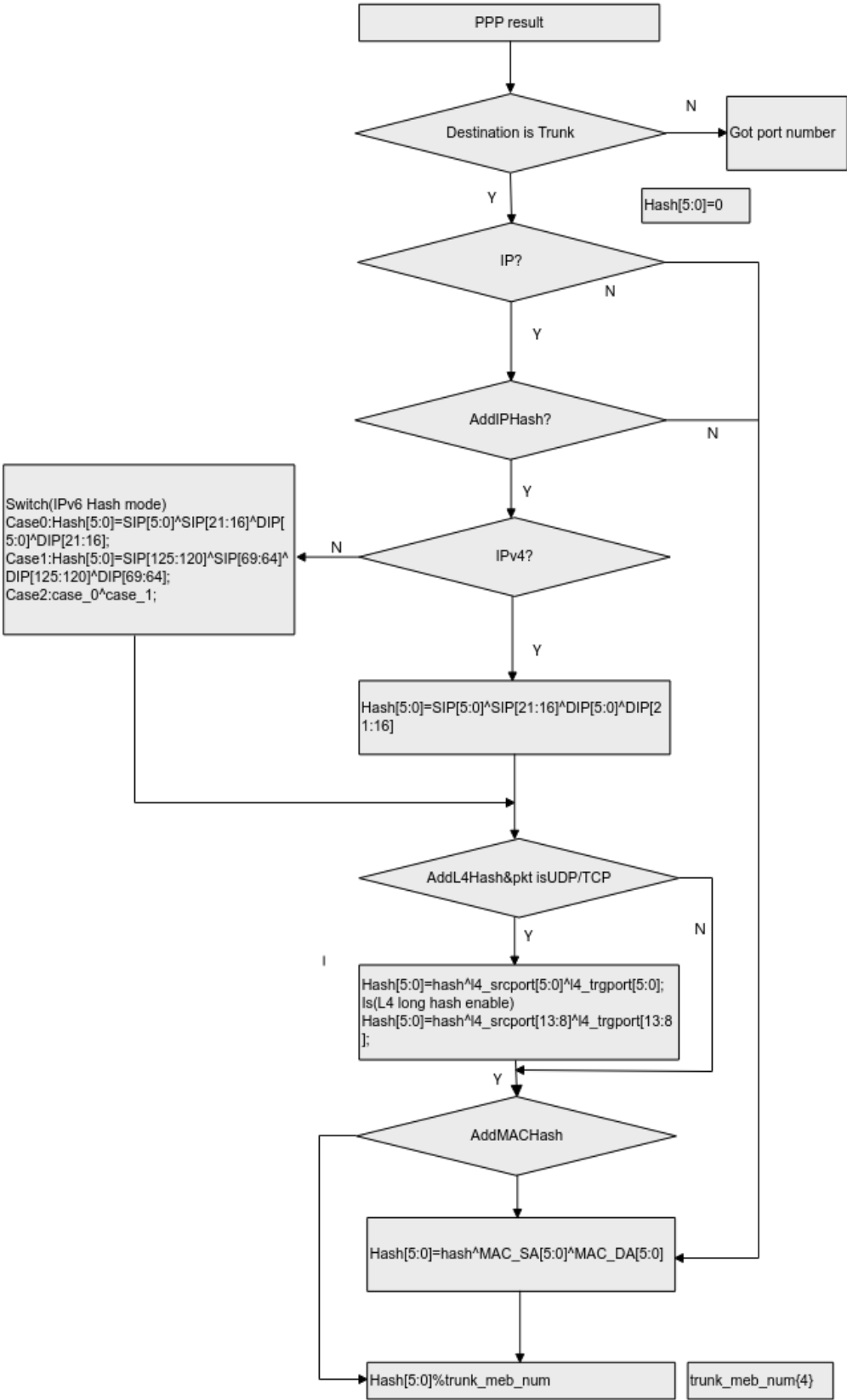
当前硬件能够同时处理IP和MAC地址的计算需求，但在实际的产品应用场景中，通常不会同时使用这两个地址。

关于Trunk选路的元素控制，全局控制如下：

- 可以选择是否让MAC地址参与计算；
- 可以选择是否让IP地址（包括IPv4和IPv6）参与计算，并且对于IPv6地址，设置三种不同的计算模式；
- 可以选择是否让UDP或TCP端口号参与计算，如果选择参与计算，还可以进一步设置long hash，将[13:8]纳入计算范围。（另外，注意在分片报文的情况下，建议不要让UDP或TCP端口参加hash运算）；
- 所有的上述元素都可以独立设置mask，对于设置了mask的元素，在计算过程中会用0代替其值进行计算。



图 4-40 简单异或算法



- **CRC32**
在CRC算法中，通常会利用一系列关键信息进行CRC计算。如表4-30所示。

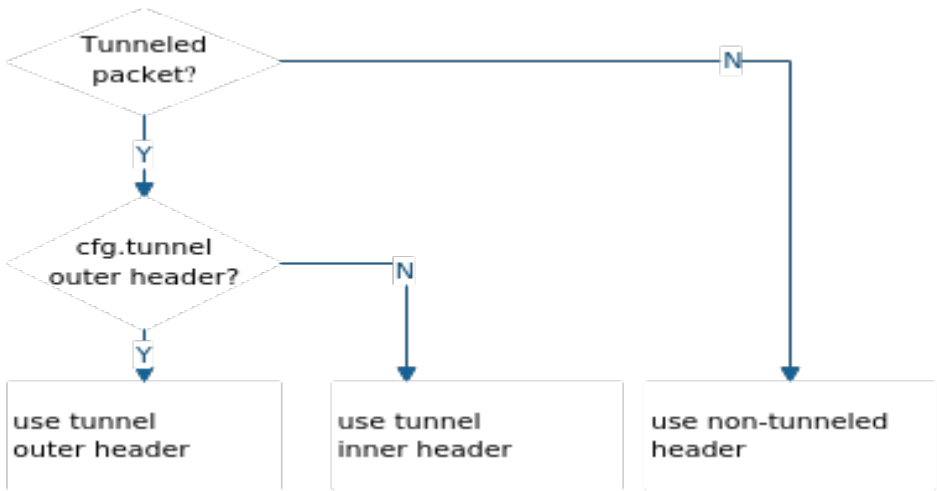


表 4-30 CRC 输入 key

域	位宽	描述
iport	8	bit[7]: 保留。 bit[6]: 0 for TX , from CPU, 1 for RX, from mac port。 bit[5:0] TX: ppe_tnl 0~25; RX: mac 0~31。
dmac	48	报文的目的MAC地址。
smac	48	报文的源MAC地址。
VLAN	16	报文查表转发的VLAN tag。
dip	128	报文的目的IP地址, IPv4时仅低32bit有效。
sip	128	报文的源IP地址, IPv4时仅低32bit有效。
dport	16	TCP/UDP的目的端口号。
sport	16	TCP/UDP的源端口号。

在隧道报文时，可以根据配置选项来决定是否采用隧道的外层头部信息还是隧道的内层头部信息来进行hash计算。

图 4-41 Trunk hash 隧道报文 key 选择

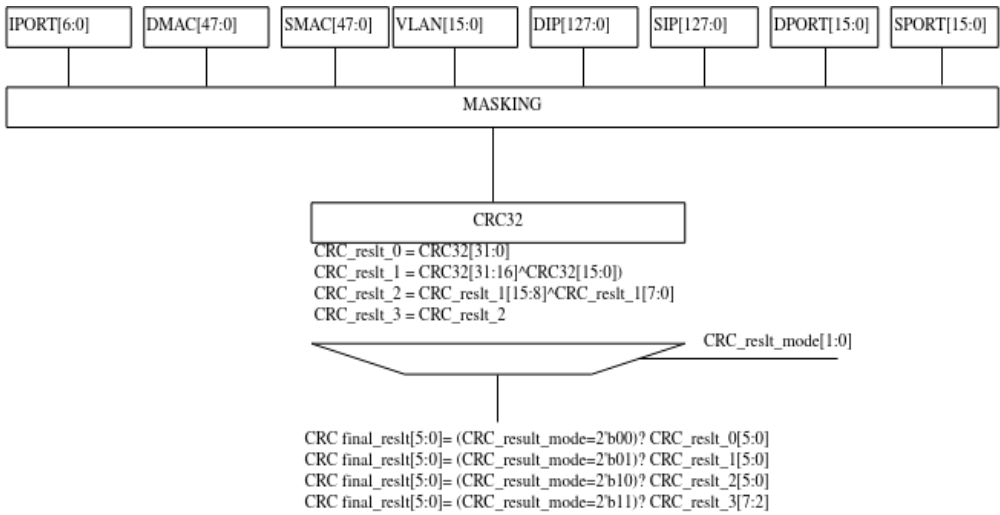


在准备完毕key后，首先会执行屏蔽操作，即排除那些不参与运算的位字段，随后将处理后的数据送入CRC32计算引擎中进行计算CRC结果，CRC32使用的多项式为：

$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ （对应的十六进制表示为0x04C11DB7）

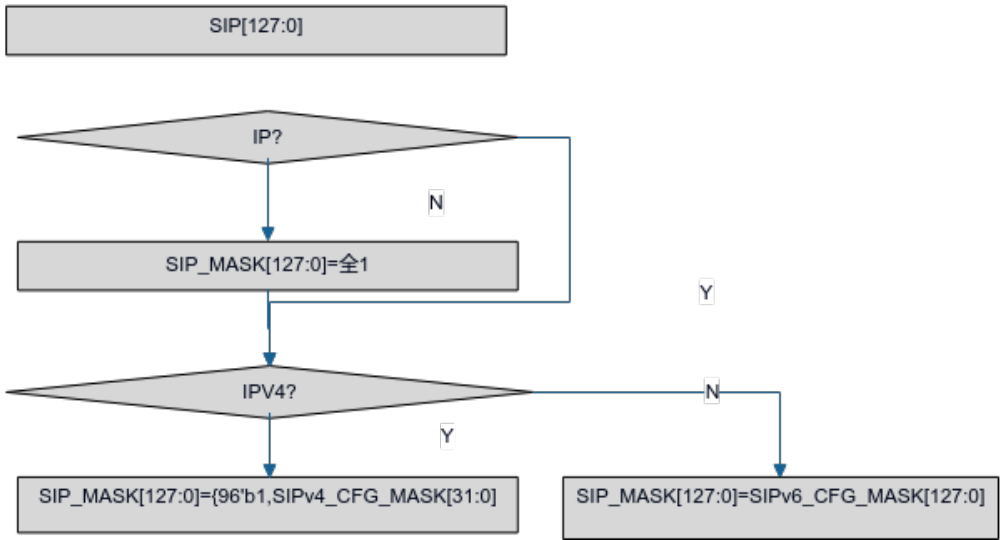
计算得到的CRC32结果是一个32位的值，但实际应用中仅使用其最低的6位，因此，在高位发生变化而低位不变的情况下，可能导致hash结果不佳，为了改善这种情况，可以根据具体需求配置对结果进行一次或者多次折叠异或操作。

图 4-42 Trunk CRC 运算



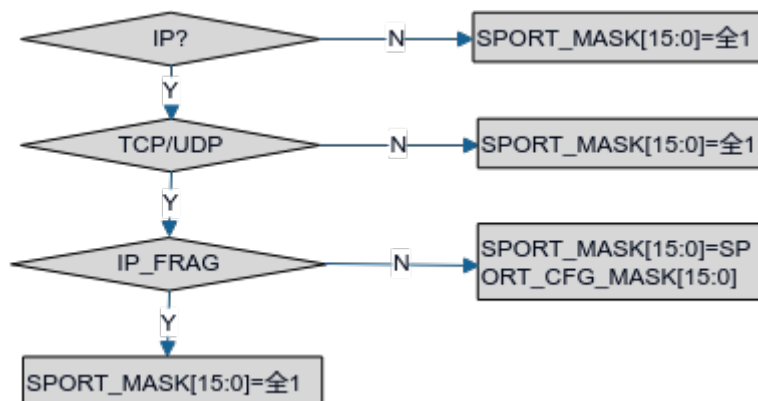
在屏蔽时，IP、SPORT和DPORT需要特殊处理，具体处理流程如图4-43所示。

图 4-43 Trunk IP 屏蔽处理



以SIP为例，在非IP报文时，SIP域段全部屏蔽，如果是IPv4报文，那么高的96bit全部屏蔽，低的32bit使用IPv4的配置值决定是否屏蔽，如果是IPv6报文，那么128bit全部使用IPv6配置值决定是否屏蔽。

图 4-44 Trunk TCP/UDP 端口号屏蔽处理



首先，确认接收到的数据包是否为IP报文。如果不是IP报文，则应将其屏蔽。如果确认是IP报文，接下来需进一步判断该报文是否属于TCP或UDP协议。若不属于这两种协议，则同样进行屏蔽处理。对于TCP或UDP报文，还需检查其是否为分片报文。由于非首片分片中的源端口（SPORT）和目的端口（DPORT）信息可能与首片不同，这可能导致同一连接内的数据包顺序混乱。因此，无论是否为分片分片，都应对SPORT和DPORT进行屏蔽操作，以防止此类问题的发生。而对于未分片的报文，仍可利用SPORT和DPORT进行哈希计算。

对于未解密的数据包而言，IP地址和端口号字段将被视为无效。

在处理VLAN标签时，仅使用实际用于转发过程中的VLAN标签信息。

4.4.5.3.7 镜像

对于镜像，支持2种镜像：

- 入口镜像：
 - 基于入口流表镜像，在流表里指示报文需要镜像。
 - 基于入物理口+通道口镜像，将某个端口的接收报文，固定镜像到某个出口。
 - 入口镜像的报文是原始收到的报文。
- 出口镜像。
 - 基于出口流表镜像，在流表里指示报文需要镜像。
 - 基于出物理口+通道口镜像，将发送到某个端口的报文，固定镜像到某个出口。
 - 出口镜像的报文是经过VLAN编辑后发出的报文。
 - LSW丢弃的报文不会做出口镜像，也不统计，分发流表丢弃的会做。
- 镜像报文上CPU时，送到软件指定的配置的逻辑队列号和通道号处理；镜像报文不查LSW出口流表，也不查分发表，只做RSS得到Queue。
- 入口和出口镜像的目的口只支持各一个。
- 出口镜像报文，VLAN层数在出口镜像目的口为DSA时不可用，Buffer type以出口原始报文的Buffer为准。

须知

出口镜像报文在做VLAN编辑时，使用镜像目的口的TPID填写。



4.4.5.3.8 VLAN 过滤表

表 4-31 VLAN 过滤表

域	位宽	描述
vlan_port	80	指示端口是否在此VLAN域。 0x0：不在； 0x1：在。 bit[51:26]：对应26个PPE_TNL； bit[25:0]：对应26个MAC网口。
vlan_tag_en	80	出口时是否需要删除vlan_tag。 0x0：删除tag； 0x1：保留tag。 bit[51:26]：对应26个PPE_TNL； bit[25:0]：对应26个MAC网口； 仅在出口时使能了端口删除使能时有效。
stp_inst	4	基于VLAN的生成树实例。

VLAN过滤表支持4K深度，共4096个VLAN ID。每个表项有对应的端口和通道号，并有Hybrid的信息。

入口的VLAN校验和出口的VLAN过滤使用同一张VLAN表。一个报文会查两次表。

4.4.5.3.9 多播复制表

须知

Hi309a需要多播统一不做补偿。

表 4-32 多播复制表

域	位宽	描述
mcast_port	52	指示端口是否需要复制报文。 0x0：不复制； 0x1：复制。 其中， bit[51:26]：对应26个PPE_TNL； bit[25:0]：对应26个MAC网口。



4.4.5.3.10 未知报文

对应LSW查表miss的报文，可以选择如下2种处理方式：

- 丢弃。
- 洪泛，查VLAN的线性表，将报文复制到除源端口的该VLAN ID的所有使能端口。

³

当unknown报文被送往CPU时，需要查询分发表以确定对应的逻辑队列号；

若采用洪泛方式处理后，报文将被视为多播/广播报文，此时应依据多播/广播机制决定是否丢弃；

在执行镜像、多播、广播、洪泛操作时，需要申请ppp内部512深度缓存资源；若在资源不足，则不进行报文复制。

说明

³：当目的端口和源端口都属于同一个Trunk组时，应进行过滤处理；若仅目的端口属于Trunk组，则需执行hash选路；对于不属于同一Trunk组的情况则不需要特别处理。

4.4.5.3.11 BPDU 报文

BPDU（Bridge Protocol Data Unit）报文可以在流表中配置为黑名单以实现过滤，并且也可以通过流表来指定BPDU报文的转发路径，BPDU报文遵循常规报文的转发规则，可能会被过滤和丢弃。由于BPDU报文参与L2表的学习过程，所以在流表中无法指定其进行L2表查询。

对于那些既未命中流表又未经过L2表处理的BPDU报文，将依据单播隔离进行过滤。

如果流表未能匹配到相应的条目，则不会进行洪泛操作，从MAC地址接收的报文会被送到CPU处理，而由CPU发出的报文则根据不同的发送端口重新发送到对应的MAC接口。

4.4.5.3.12 LSW 直通报文

LSW通过在发送描述符中的SC域来决定是否绕过LSW的查表过程，并使用SC_TARGET指定直接发送的目标位置，这可以是环回，不同ppe_tnl之间和MAC口。这类报文将不进行任何修改操作，包括VLAN卸载，而是直通输送至目的位置。然而，为了增强灵活性，系统支持基于源端口的VLAN过滤的使能配置。同样支持端口隔离。

当SC域设置为0b10时，出口流表必须确保不将流量重定向至MAC端口。

4.4.5.3.13 端口隔离

在实现单播、多播、广播和洪泛的支持时，每种类型都需要维护独立的端口隔离表。

对于普通环回功能，目的端口列表里必须包含原端口。

配置表项时，请确保数据包的目的地址不是未使用的端口；初始配置应将所有端口设为有效（例如配置为1），以避免因无效设置导致的数据包被过滤。



4.4.5.3.14 转发流程

图 4-45 LSW 入口处理流程

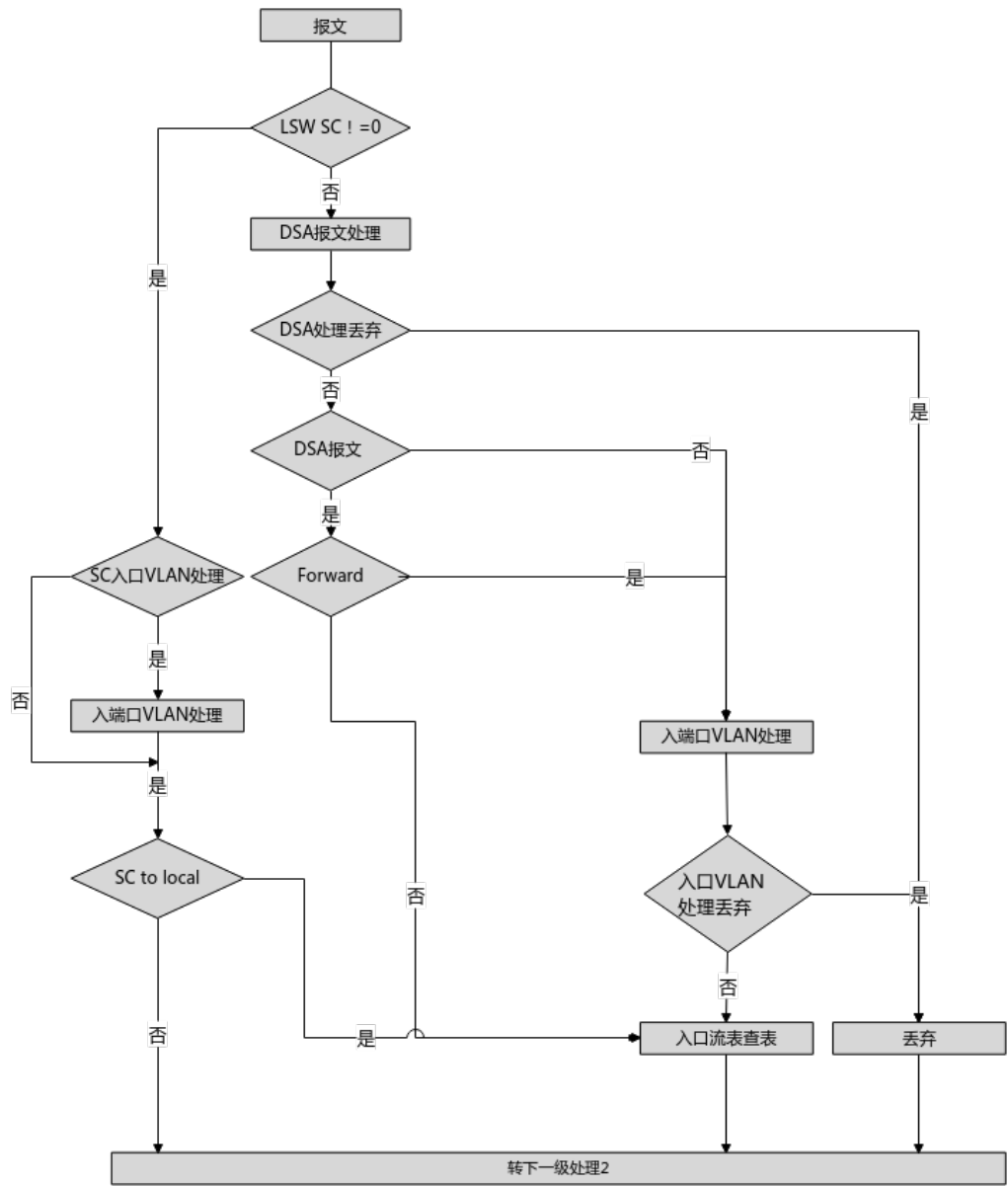
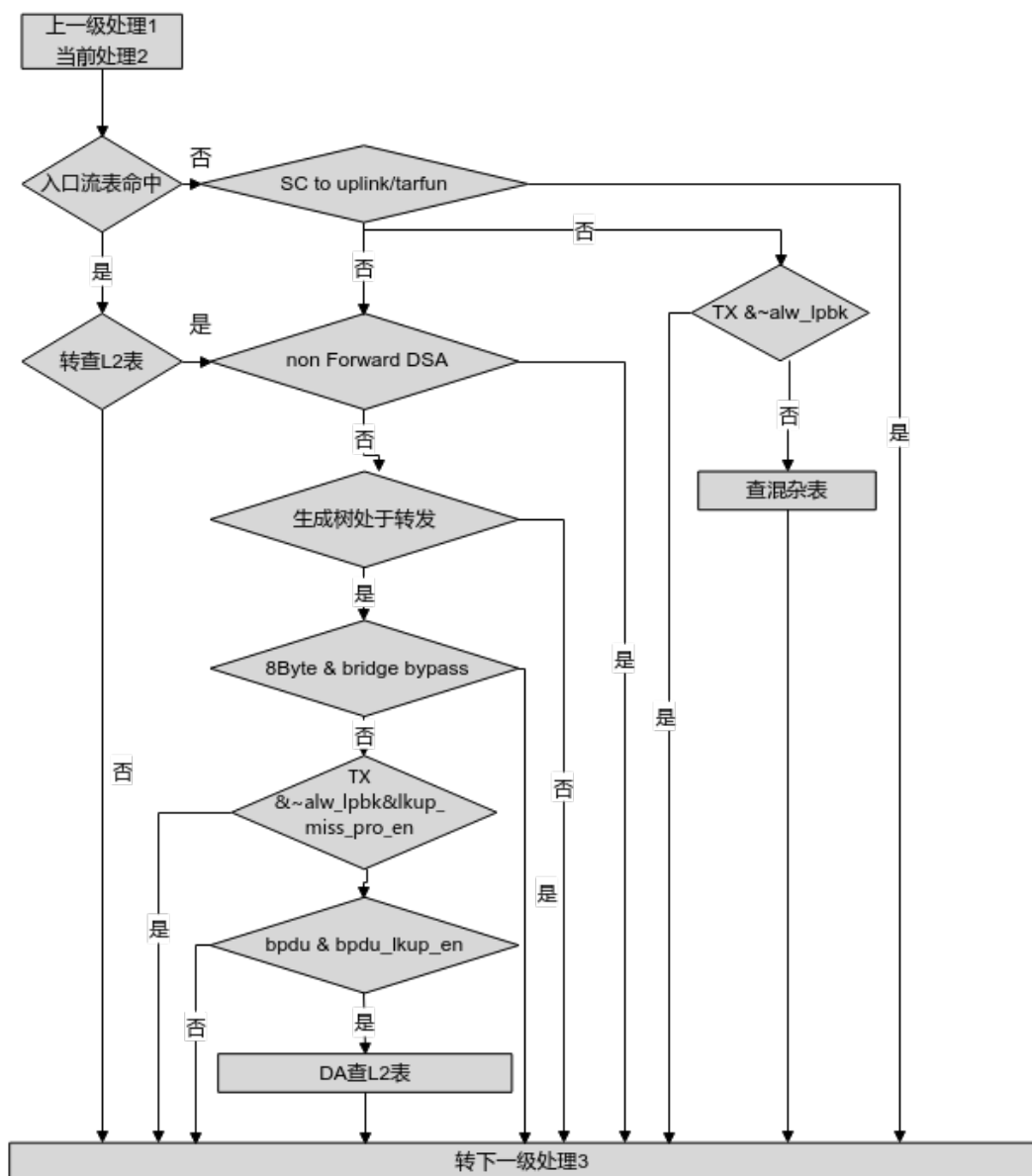


图 4-46 LSW 查 L2 表流程



如果存在SMAC查找，则使用SMAC进行L2表查询。

图 4-47 使用 SMAC 查找 L2 表路程

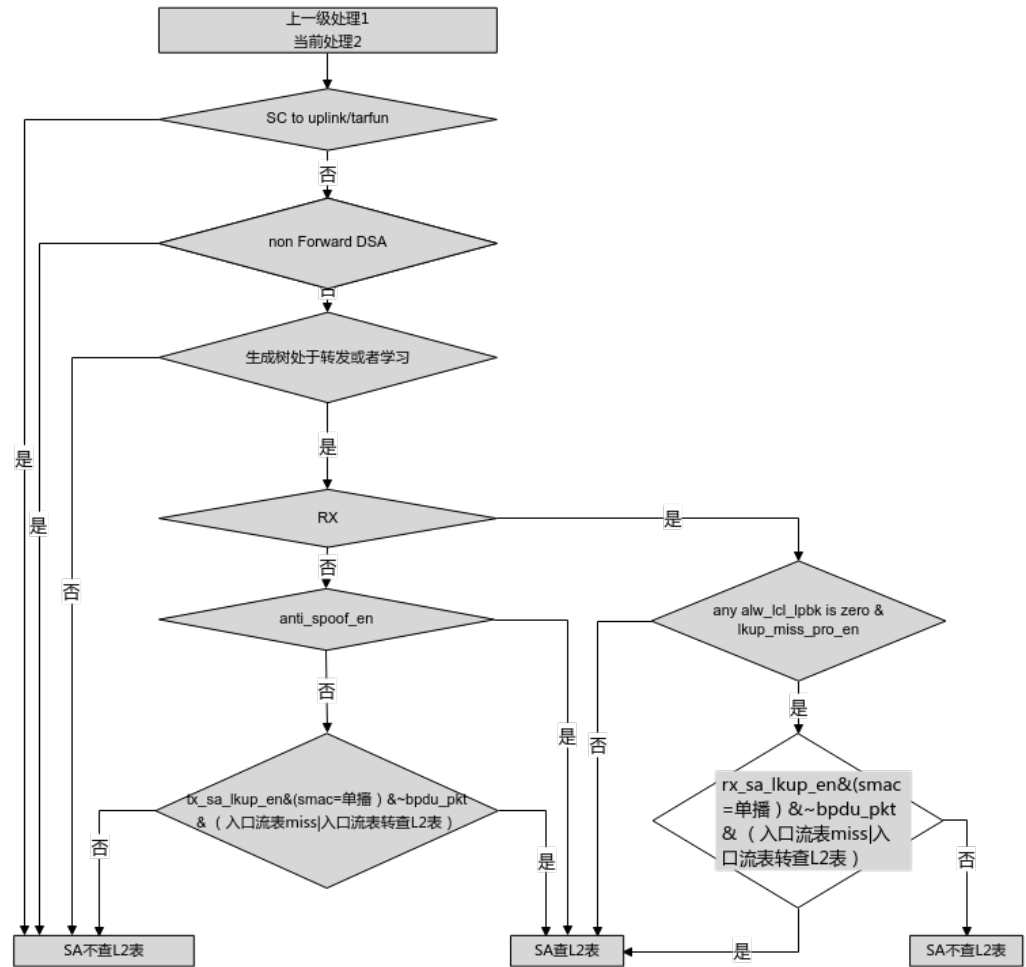


图 4-48 LSW 转发流程

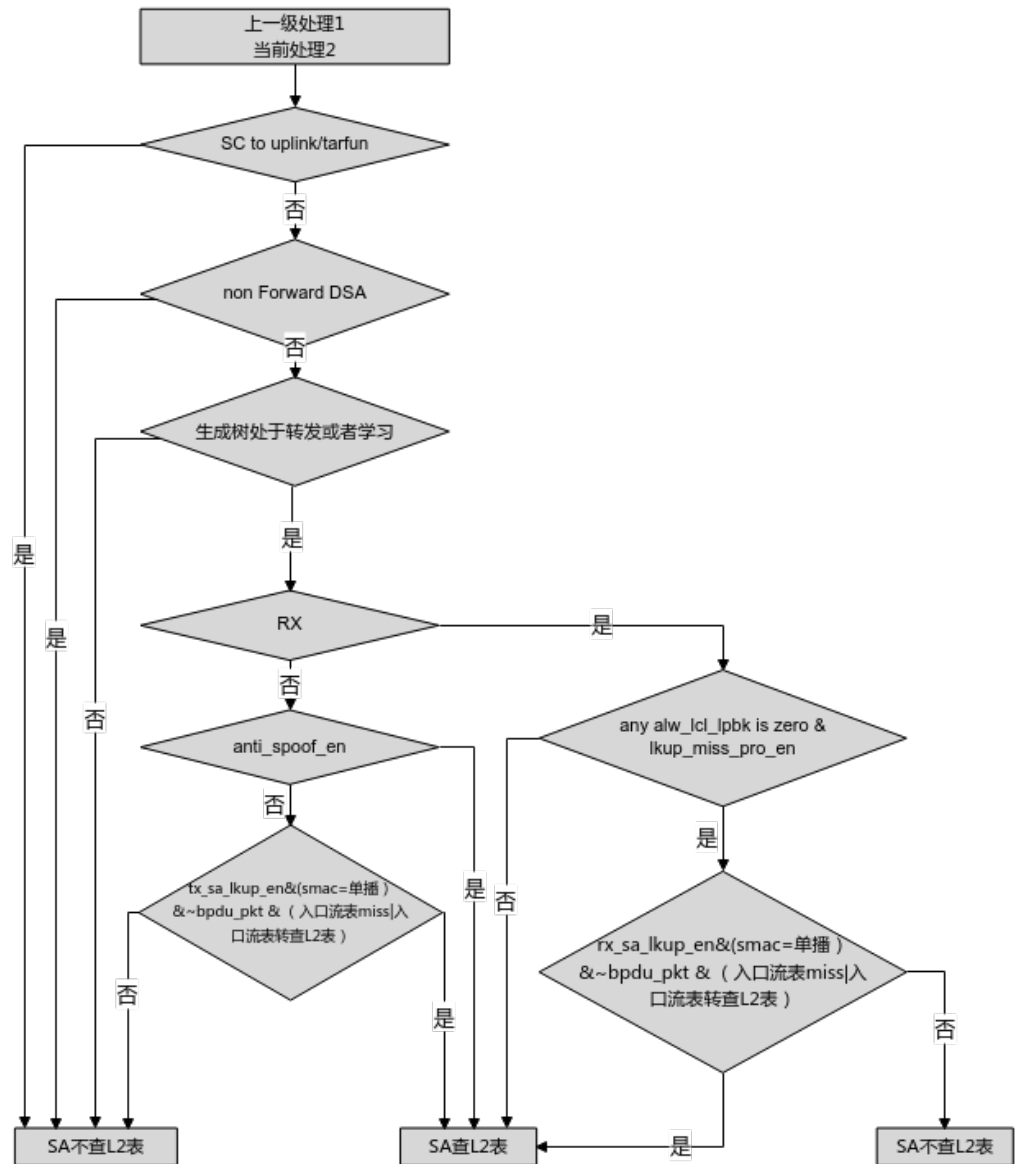


图 4-49 LSW 过滤流程

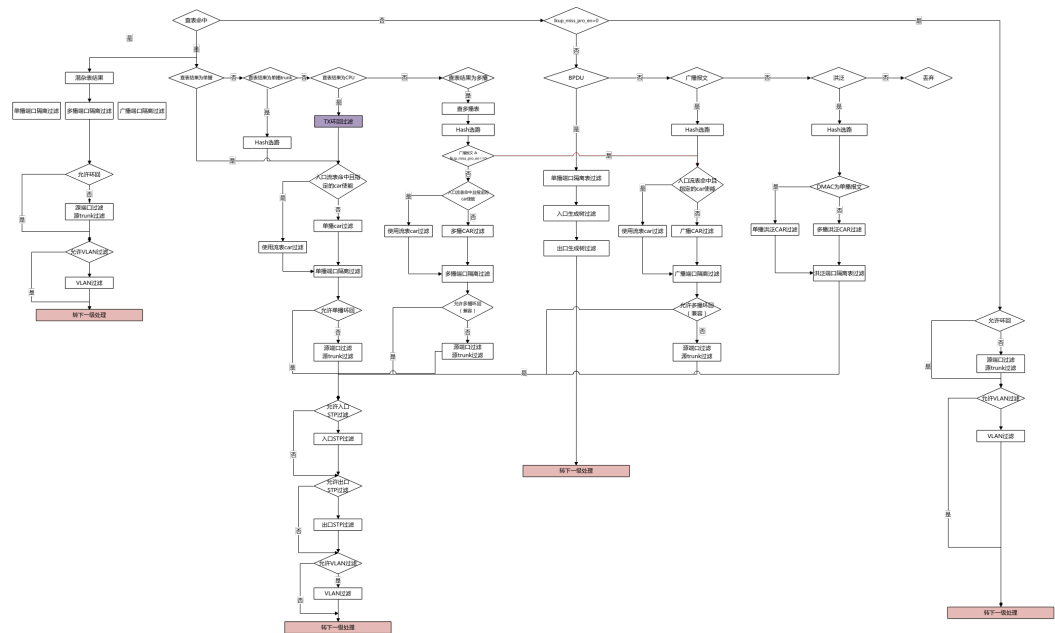
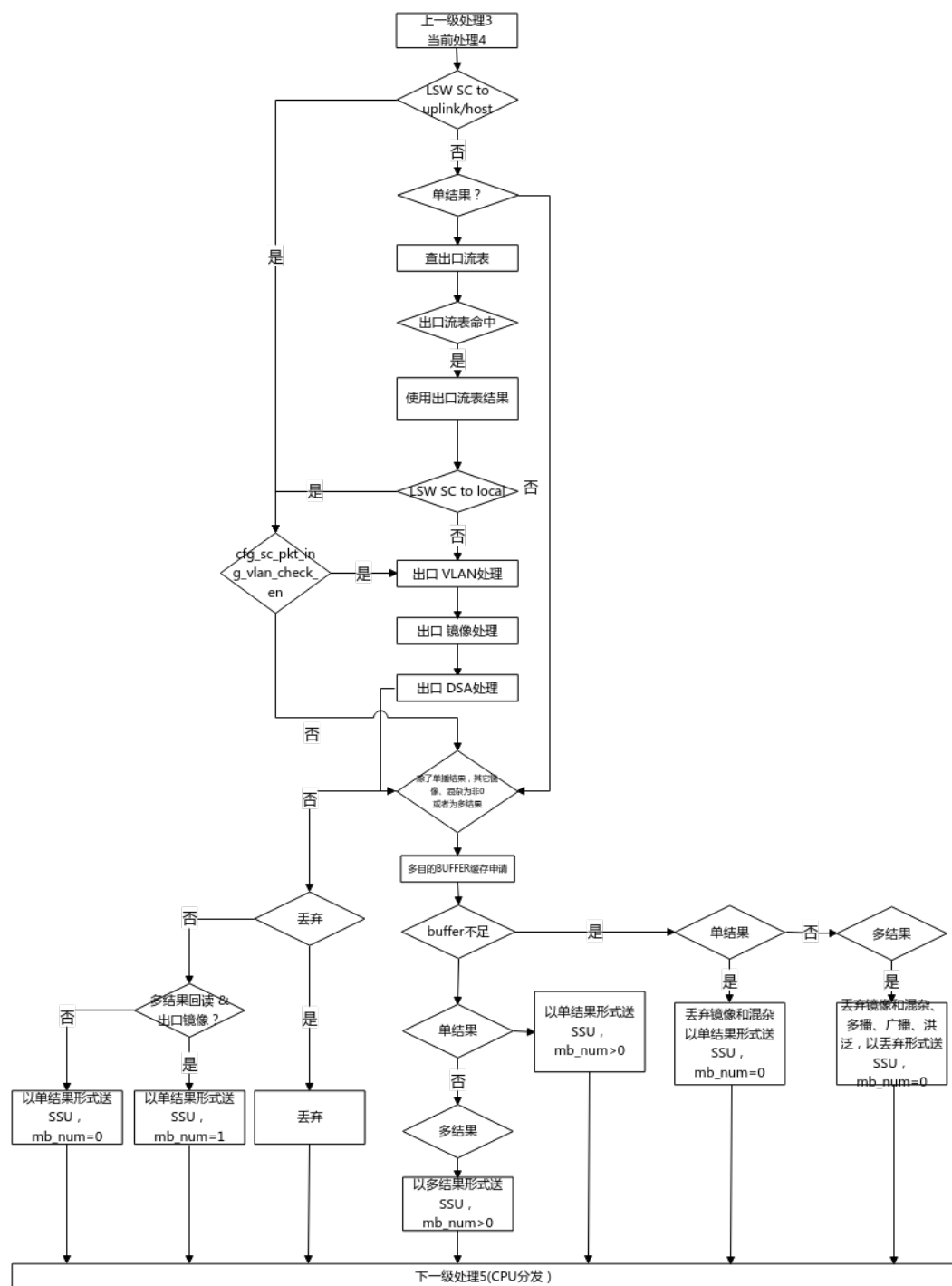


图 4-50 LSW 出口处理流程

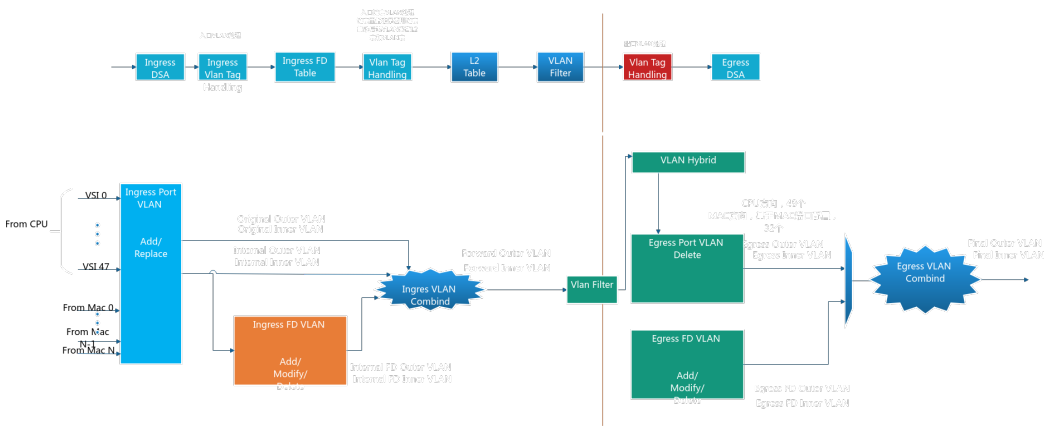


多目的的查表结果从PPP的B0口送给SSU，SSU再通过PPP的R接口进行回读。



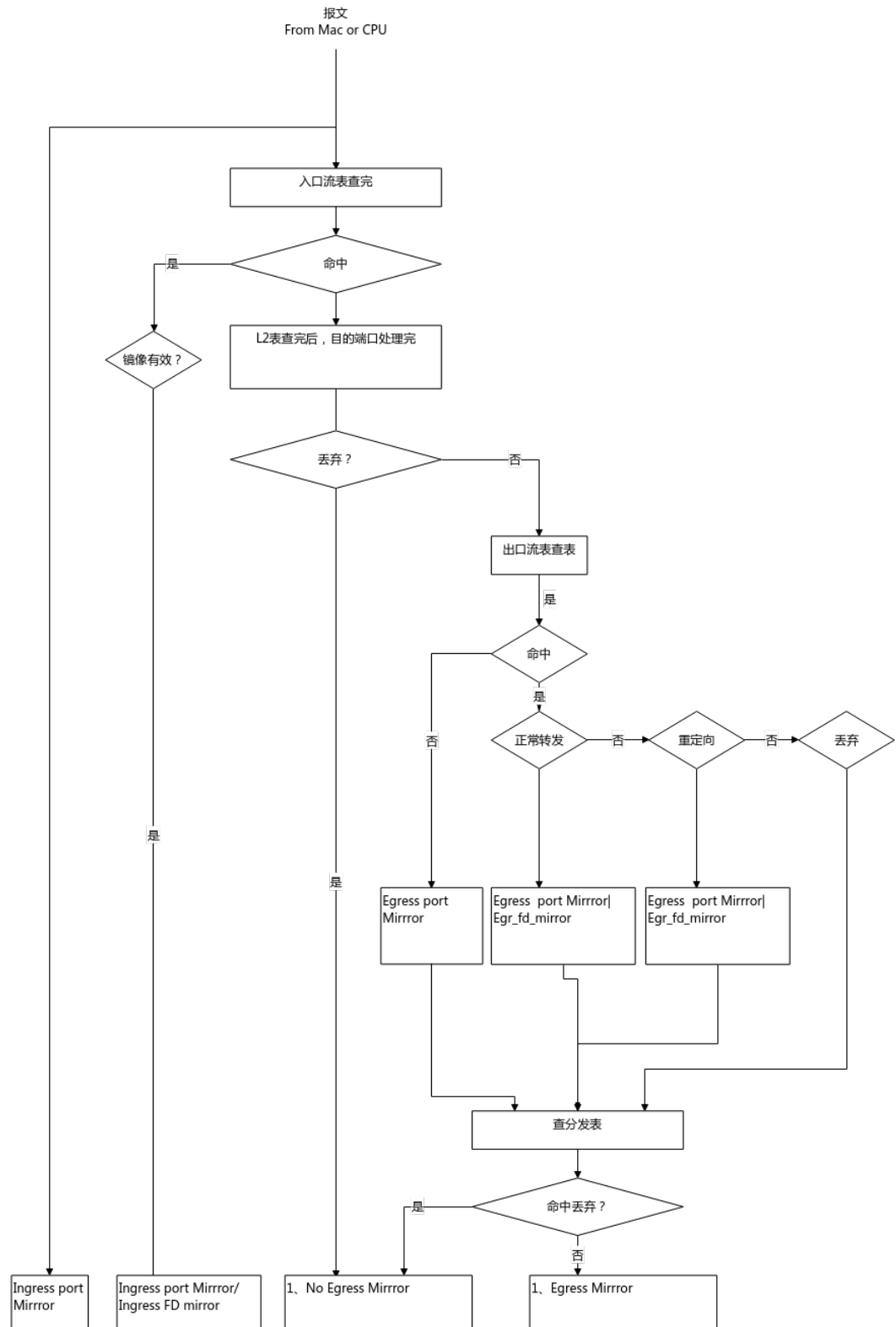
4.4.5.3.15 VLAN 卸载流程

图 4-51 VLAN 卸载流程



4.4.5.3.16 镜像流程

图 4-52 镜像流程





4.4.5.4 RSS 功能

根据报文类型选择不同的key，选择key的方式如下：

- 对于非capwap数据明文，按照如下表格的方式选择key。
- 对于capwap数据明文，根据隧道内的报文格式，按照如下表格选择key。
- 对于非IP报文，使用{SMAC, vlan1, vlan2, eth_type}，其中vlan1为靠近SMAC的，vlan2为远离SMAC的（均为隧道外的）；无VLAN按照0来进行运算。
- 根据如下公式，进行hash得到32bit，取32bit中的低8bit作为最终的fw_tnl号。

Hash方式为如下Generator2：

$$x^{32} + p_{31} * x^{31} + p_{30} * x^{30} + p_{29} * x^{29} + ... + p_r * x^r + ... + p_2 * x^2 + p_1 * x^1 + p_0 * 1$$

This is a configurable CRC polynomial. The user can configure it during the initialization.

表 4-33 RSS Key 的选择

RSS_TYPE	keys
NonF IPv4, UDP	IP4-S, IP4-D, UDP-S, UDP-D, Protocol
NonF IPv4, TCP	IP4-S, IP4-D, TCP-S, TCP-D, Protocol
NonF IPv4, SCTP	IP4-S, IP4-D, SCTP-S, SCTP-D, SCTP Verification-Tag, Protocol
NonF IPv4, Other	IP4-S, IP4-D, Protocol
Frag IPv4	IP4-S, IP4-D, Protocol
NonF IPv6, UDP	IP6-S, IP6-D, UDP-S, UDP-D, Next Header, Flow Label
NonF IPv6, TCP	IP6-S, IP6-D, TCP-S, TCP-D, Next Header, Flow Label
NonF IPv6, SCTP	IP6-S, IP6-D, SCTP-Verification-Tag, Next Header, Flow Label
NonF IPv6, Other	IP6-S, IP6-D, Next Header, Flow Label
Frag IPv6	IP6-S, IP6-D, Next Header, Flow Label
Geneve OAM	Source Outer UDP Port, VNI
VXLAN-GPE OAM	Source Outer UDP Port, VNI
非IP报文	SMAC, VLAN1, VLAN2, Eth-type



说明

- 有效IP报文的判断标准如下。
 - CPU发送方向的报文被视为有效。
 - 接收方向pkt_crypt_dec字段为1（表示已解密），也视为有效。
- 对于隧道报文，若其不是CAPWAP数据明文类型，则一定为非分片报文。
- UDP类型的判断依据如下。
 - 对于非隧道报文，若in_l4_id字段为0，则该报文为UDP类型。
 - 对于隧道报文，若ot_l4_id字段的值为1、3、4、5或6时，该报文为UDP类型，否则该报文为非UDP类型。
- 关于Flow Label字段的取值规则如下。
 - 当报文类型为CAPWAP数据明文类型或非隧道报文时，Flow Label字段的值应为gro_field的bit[19:0]。
 - 在其他情况下，Flow Label字段的值应设为0。

4.4.6 内部报文缓存管理

4.4.6.1 流控功能

4.4.6.1.1 普通流控

基于物理接口在PPE内部所使用的资源进行流量控制和解除留流控制，并允许单独配置每条链路的流量控制阈值（水线）。

当某个Pool的Buffer不够时，系统能够实施流量控制。

具备超时机制：如果接收到数据包后，且BMU的Buffer大小低于设定的水线值，在特定的时间段内未恢复，则对发送方实施流量控制；一旦超出该时间段，自动解除流量控制。

每个端口都拥有独立的缓存资源分配，确保不同端口间的资源互不影响，避免因资源竞争导致的通道阻塞问题。

同时，每个端口的流量控制阈值需能有效应对网络中可能出现的数据包突发情况。

4.4.6.2 队列管理

缓存与队列管理

- 缓存空间：总共提供1.5MB的包缓存空间，基于Cell进行管理，每个Cell大小为256Byte。所有出口队列共享6144个Cell，这些Cell被划分为3个Bank，每个Bank包含2048个Cell。

列队分配

- MAC转发：每个端口有4个队列，通过PPP查表获取PRI的低2位做为队列号。
- RPU/RoCE RX转发：每个TNL有4个出口队列，RPU和RoCE RX相同的TNL共享接收队列。

调度机制

- MAC转发列队：使用ETS进行调度（ETS调度介绍详见“[ETS配置](#)”）。



- 转发到CPU的列队：使用SP（严格优先级）/WRR（加权循环）调度。
- MAC与RPU/RoCE RX之间的调度：使用RR（轮询）调度。

丢弃策略

- MAC转发：支持基于端口进行的Cell和PD粒度设置丢弃水线，当达到丢弃条件时，执行尾丢弃操作。对于多播复制量，也支持基于端口的Cell和PD粒度设置丢弃水线，当达到丢弃条件时，对相应端口的多播报文执行尾丢弃。
- RPU/RoCE RX转发：支持基于Cell和PD粒度设置丢弃水线，当达到丢弃条件时，对相应TNL的多播报文执行尾丢弃。

4.4.6.3 典型场景 Buffer 和水线配置

4.4.6.3.1 多播 FIFO 配置

启动单播强制高优先级模式，并将权重偏移量设置为6。

表 4-34 多播 FIFO 配置

-	大小	调度比例	Shaping（默认）
高优先级多播	256	1	150G
低优先级多播	256	2	300G
低优先级单播	508	2	420G

4.4.6.3.2 队列配置

每个端口可配置对应的Cell与PD丢弃阈值。

每个队列可配置对应的Cell与PD丢弃水线配置。

每个队列预留PD，在数据入队时优先占用预留的PD；在数据出队时优先释放共享PD。

配置防止OQ溢出的保护阈值，当总流量达到该阈值时，入口处开始丢弃接收数据包，并对发送方向施加压力，同时调整多播和镜像调度策略。

4.4.7 BMU、POE 交互

4.4.7.1 BMU Buffer 申请与释放

MAM模块负责向BMU申请Buffer，并依据BMU的QoS反馈机制处理RPU方向上的Buffer请求。同时，它还接收TPU方向上的Buffer释放请求，并将释放的Buffer回传给BMU，此外，MAM支持CPU对Buffer的直接读写操作。

MAM内部配置了16个独立的Pool，每个Pool可以单独设置不同的水位线。

当某个Pool中的Buffer不足且达到预设的低水位线时，MAM将向BMU申请额外的Buffer；一旦从BMU获取到新的Buffer，他们会被写入相应的Pool中。

在决定是否从FIFO（先进先出列队）中读取释放的Buffer时，MAM会综合考量Pool中的Buffer数量是否需要本地释放标志，以及当前未完成的outstanding情况。如果条件

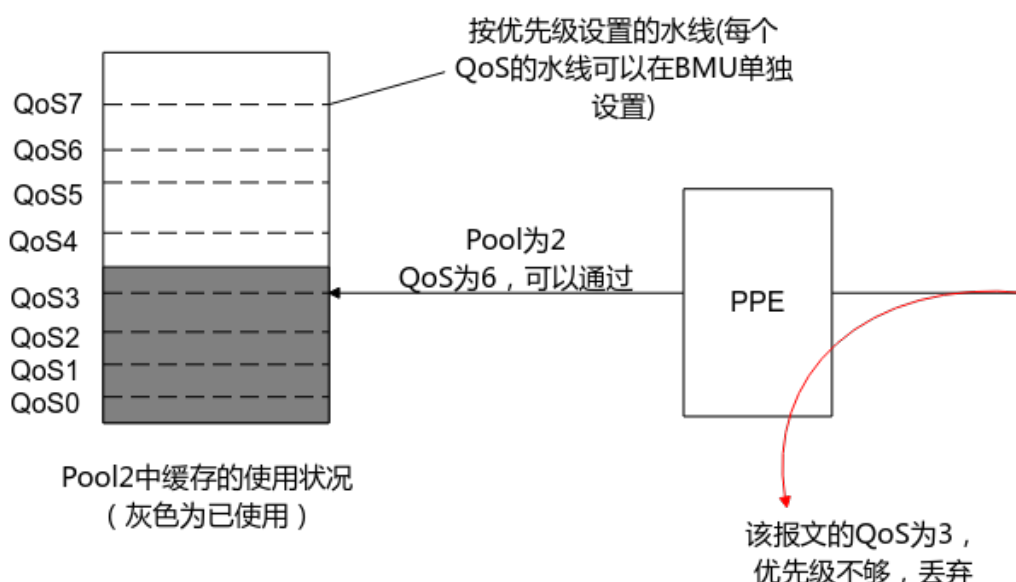
允许，MAM将从FIFO中读取Buffer并决定是将其回传给BMU还是保留在本地的Pool中。

4.4.7.2 BMU QoS 丢弃报文

当系统的总Buffer不足时，如果端口继续接收数据包，会导致所有的Buffer被占满，进而使得后续的数据包无法被处理而被丢弃，这将严格影响业务运行。为了解决这个问题，PPE采用以下策略：

首先对接收到的数据包进行解析，提取出VQI、QoS以及数据包大小等信息，然后根据VQI确定所需要使用缓存的Pool号，接着检查该Pool在当前BMU的丢弃阈值状态，最后依据数据包的优先级来判断是否应该将其丢弃，这种机制能够有效避免因Buffer溢出而导致的数据包丢弃问题。

图 4-53 按照 QoS 丢弃报文



须知

每个QoS的水线都可以在BMU单独设置，没有限制QoS7的水线一定要比QoS0的大或小。

4.4.7.3 POE 消息交互

在接收方向完成报文缓存的过程中，需要将报文任务通过credit机制推送给POE，以确保报文顺序正确，具体来说，PPE将credit计数值附加到每个报文任务的SN号域段中。

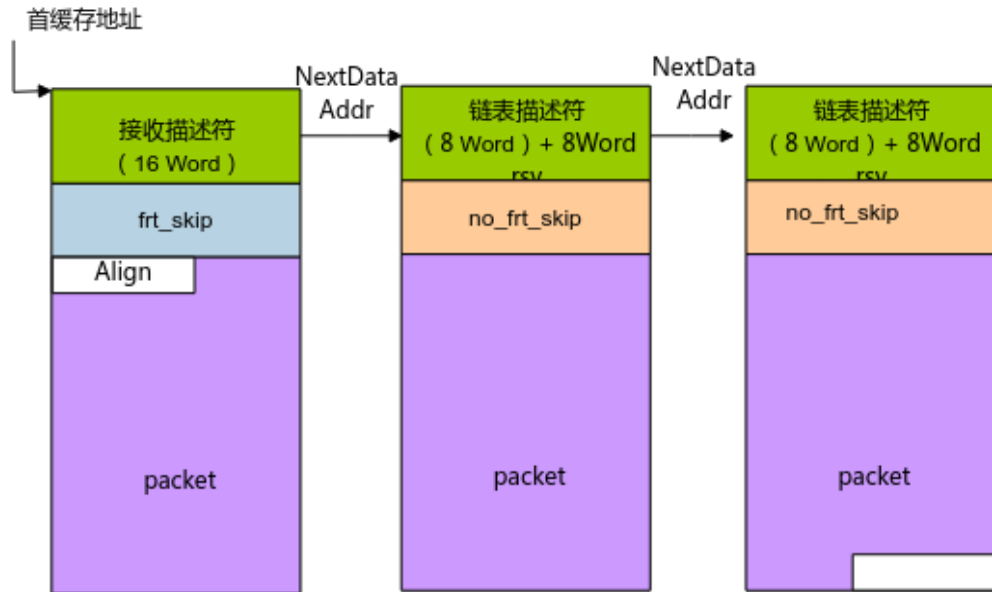
所有credit是循环使用的，当没有credit时，PPE将不会添加新的任务，如果POE因为队列已满（包括真满、假满）超时或丢弃等原因无法处理新任务，则会释放credit给PPE，此外，PPE还设置了对POE的Outstanding任务数量进行控制，以此来管理总线带宽。

4.4.8 接收通路

4.4.8.1 支持的接收报文存储方式

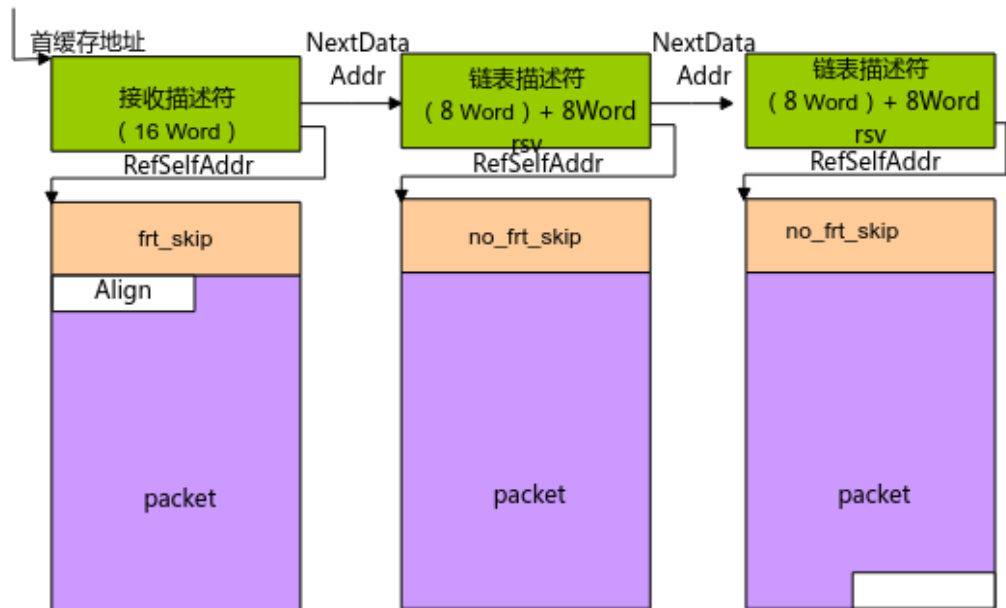
当接收到的报文无法由单一Buffer容纳时，会采用多个Buffer进行存储。如图4-54所示。

图 4-54 接收报文在 DDR 中的存储方式示意图（描述符报文非分离模式）



当分发表项设置了描述符和报文分离后，数据的存储方式如图4-55所示。

图 4-55 接收报文在 DDR 中的存储方式示意图（描述符报文分离模式）





须知

如果软件错误配置某个Pool中skip和buf_size的大小关系，有可能导致buf出现泄露，需要软件保证配置的正确性。此时PPE会上报中断至对应VF。错误配置包括如下情况：

- 分离情况下，MAM中配置的cfg_buf_size<skip+1，单位64Byte。
- 非分离情况下，MAM中配置的cfg_buf_size<skip+2，单位64Byte。

4.4.8.2 Write allocate 功能

在使用FW_TNL设置描述符和报文头或将完整报文写入L3以降低CPU访问延迟并提高性能时，Write Allocate操作可通过访问总线的Force Write Allocate指示进行控制。

4.4.9 发送通路

在处理报文时，软件首先构造好报文以及相应的描述符，并将描述符的地址推送至PPE（Packet Processing Engine），PPE随后执行以下操作：固定计算IPv4 checksum并更新checksum域、根据指示进行UDP/TCP checksum计算和更新checksum域。

- 对于Y.1731报文，软件可以指示在特定位置插入流统计信息和时间戳。
- 而对于PTP（Precision Time Protocol）报文，PPE则会指示MAC层进行时间戳的添加，或者修改修正域或通过中断上报回传时间T3、sequence ID。

4.4.9.1 支持的发送报文存储方式

在发送报文时，支持以下两种方式：

描述符与报文共享一Buffer；

描述符与报文使用不同的Buffer。。

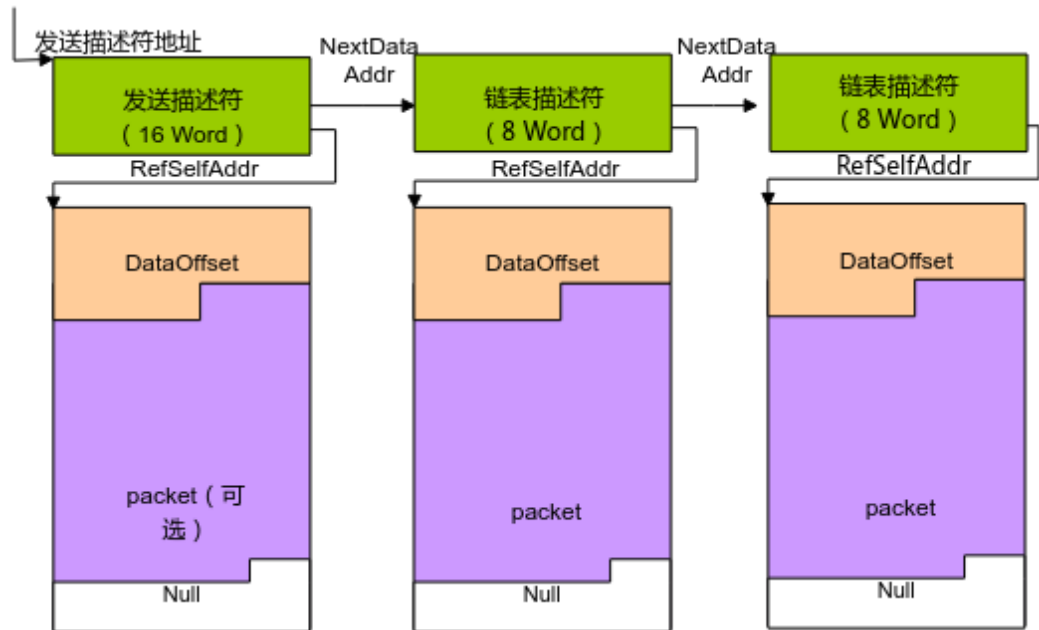
判断方法：通过对比报文的描述符地址和RefSelfAddr是否一致来确定，若两者地址相同，则表示描述符和报文共用一个Buffer，反之，则表明他们各自使用独立的Buffer。

如果首片的RefSelfAddr值为0，则表示首片中不含实际报文数据，但后续的分片必须包含有效的报文内容。

须知

BMU支持将首分片的RefSelfAddr设为0的Buffer正常释放，而需要先配置相应的寄存器。在启动这一功能之前，请确保已正确设置BMU相关的寄存器，以保证Buffer能够正常释放。

图 4-56 发送报文在 DDR 中存储方式



4.4.9.2 报文长度约束

系统最小支持15Byte的报文长度，小于15Byte的报文会被硬件直接丢弃。此外，系统支持通过PPE_TNL使能发送填充功能，当此功能启用时，TPU会将报文填充到60Byte，然后MAC模块会添加4Byte的CRC后发送出去。

表 4-35 报文长度约束

是否开启padding	报文大小	处理结果
Enable	<15B	丢弃。
	>=15B	不丢弃，padding到60B。
Disable	<33B	丢弃。
	>=33B	不丢弃，不做padding。

4.4.10 Telemetry 功能

4.4.10.1 INT/IOAM 业务流程

系统支持两种不同的Telemetry协议：分别是In-band Network Telemetry (INT) 和 In-situ Operations Administration Maintenance (IOAM)。PPE能够解析携带INT/IOAM info信息的数据包，并且支持解析格式为IPv6、GRE、TCP/UDP、VxLAN-GPE等格式的数据包。然后，对于采用Geneve封装的Telemetry数据包，由于Telemetry信息位于Option字段内，PPE不会对其进行特别处理。INT header、IOAM header的具体格式分别如[图4-57](#)、[图4-58](#)所示。



图 4-57 INT header

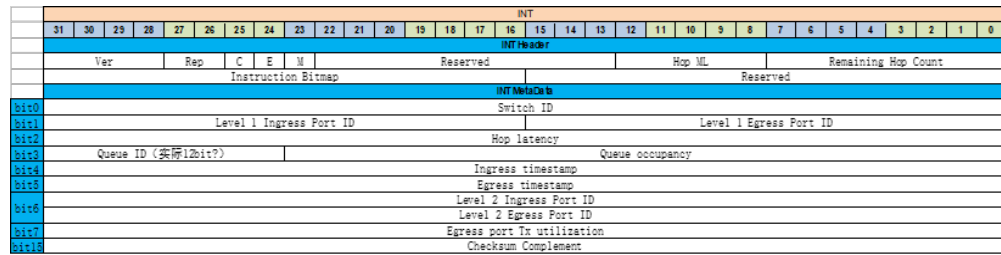
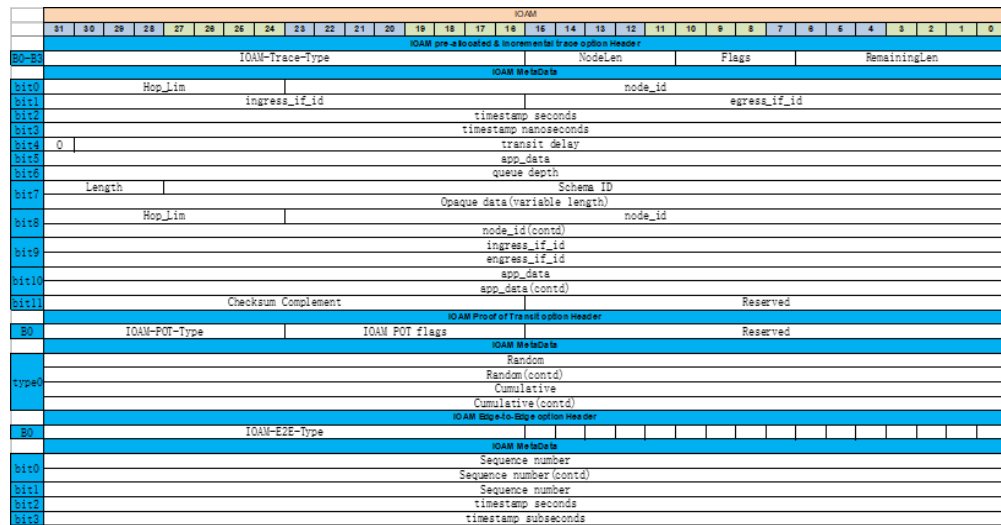
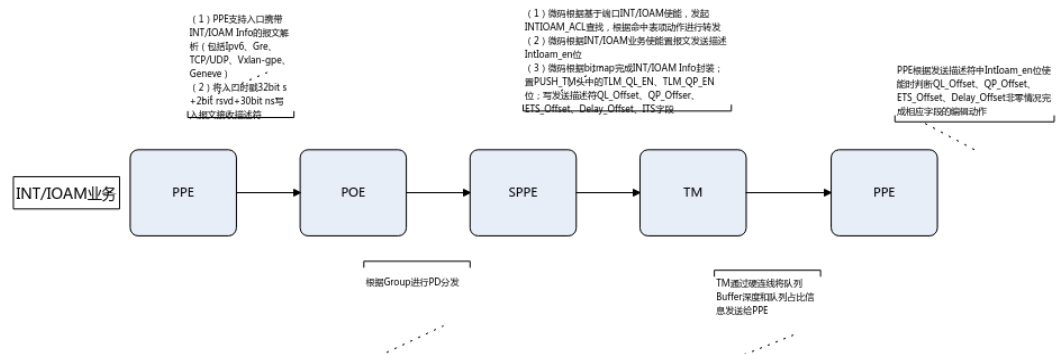


图 4-58 IOAM header 的三种格式



INT/IOAM业务流程如图4-59所示。

图 4-59 INT/IOAM 业务流程图



4.4.10.2 报文接收

在接收端，PPE（Packet Processing Engine）会识别并跳过INT/IOAM Header，然后继续解析后续的报文信息。INT/IOAM Header会封装在以下两种场景中：



- 对于非隧道报文，INT/IOAM Header可能被封装在IPv6的Hop-by-hop，Destination两种扩展头内或者GRE、TCP/UDP头之后。
- 对于隧道报文，INT/IOAM Header则会被封装在VxLAN-GPE、Geneve隧道头之后。

对于具体的IPv6报文，INT/IOAM Header的封装格式分别如表4-36和表4-37所示，PPE在处理这些Header时，会像普通的IPv6扩展头那样进行，不会做额外的特殊处理。

表 4-36 INT/IOAM over IPv6 Hop-by-Hop Extension Header

31	24	23	16	15	8	7	0
IPv6 Header							
Next Header		Header Length		Hop-by-Hop Addresses			
INT/IOAM Header							
L4 header/payload							
.....							

表 4-37 INT/IOAM over IPv6 Destination Options Extension Header

31	24	23	16	15	8	7	0
IPv6 Header							
Next Header		Header Length		Options			
INT/IOAM Header							
L4 header/payload							
.....							

在普通GRE、TCP/UDP报文中封装INT/IOAM Header的具体格式如表4-38所示。封装后的报文经过PPE时，PPE会像处理普通的GRE、TCP/UDP报文一样进行处理，不会做任何特殊操作。

表 4-38 GRE、TCP/UDP 报文中封装 INT/IOAM Heade

31	24	23	16	15	8	7	0
GRE, TCP/UDP							
INT/IOAM Header							
L5 header /L7 header/payload							
.....							



当在隧道报文中使用INT/IOAM Header时，其具体格式如图4-60和图4-61所示。PPE通过VxLAN-GPE头部的Next_protocol字段来识别是否包含INT/IOAM Header，并且该识别值是可以配置的。此外，PPE还通过INT Header前的32位指示位中的Length字段获取INT/IOAM Header的长度，并跳过INT/IOAM Header以继续解析隧道内的报文。

图 4-60 INT over Vxlan-gpe Header (INT header 在 IOAM Option and Data Space)

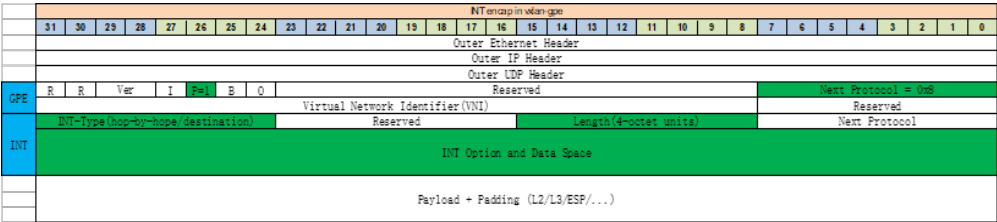
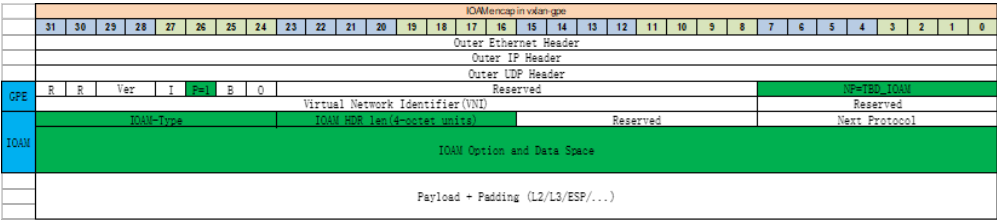


图 4-61 IOAM over Vxlan-gpe Header (IOAM header 在 IOAM Option and Data Space)



在处理隧道报文时，INT/IOAM Header在Geneve封装格式中的位置如图4-62、图4-63所示。PPE设备依据Geneve头部中的protocol_type字段来识别是否包含INT/IOAM Header，该识别值可以配置。此外，PPE还通过INT Header前的32位指示位中的Length字段来确定INT/IOAM Header的具体长度，并据此解析隧道内部的报文。

须知

在处理VXLAN-GPE封装的Telemetry IOAM数据时，PPE PA需要从IOAM头部提取HDR_LEN字段。通过将VXLAN-GPE隧道头部的位置与HDR_LEN相加，可以跳过Telemetry 头部并直接解析隧道内部的信息。需要注意HDR_LEN包含了4Byte的SHIM头部（包括IOAM-Type，IOAM HDR LEN，Next Protocol）和IOAM Option and Data space。

图 4-62 INT over Geneve Header (INT header 在 IOAM Option and Data Space)

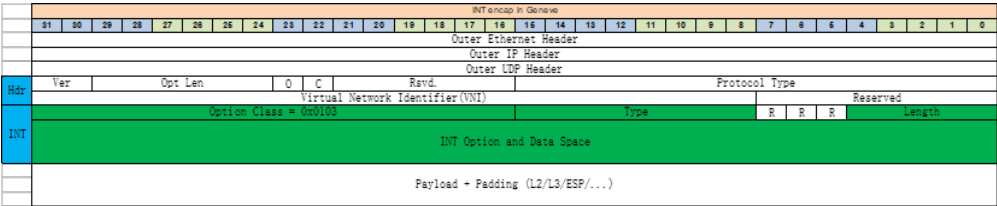
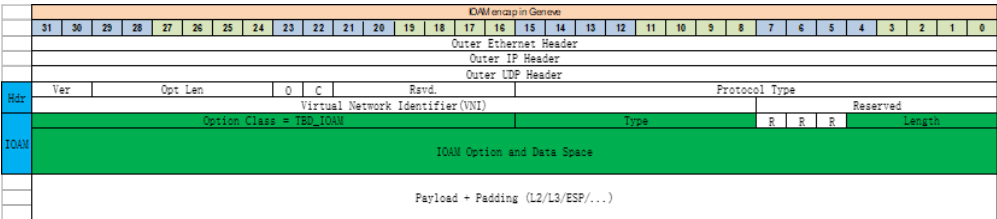




图 4-63 IOAM over Geneve Header (IOAM header 在 IOAM Option and Data Space)



除了分片报文（包含IP分片和TCP分片），PPE会对所有其他报文记录其进入时间，并将时间写入接收描述符中。时间信息格式为：{32位秒数据，2位原始秒数据，30位纳秒数据}。

4.4.10.3 发送方向

在发送方向，PPE会基于报文中的相关信息执行发送时戳的添加及发送与接收时间差的计算等操作，并在此基础上重新计算L3/L4 checksum。这些信息将随着报文一起传输，并在传输管理（TM）发送信息或发送描述符中体现。

图 4-64 发送方向处理字段

字段	位宽	说明
QL_Offset	8	Queue Length Offset, 队列深度, 编辑4Byte; 0则不编辑
QP_Offset	8	Queue Occupancy Percentage Offset, 队列占用率, 编辑4Byte; 0则不编辑
ETS_Offset	8	Egress Timestamp Offset, 出口时戳偏移, 编辑4Byte; 0则不编辑; 2bit second + 30bit Nanosecond
Delay_Offset	8	Transit Delay Offset, 时延偏移, 1 bit overflow指示位 + 31 bit 时延 (ns), 编辑4Byte, 如果超2^31-1, 则0x80000000; 0则不编辑
ITS	32	Ingress Timestamp, 入口时戳, 2bit second + 30bit Nanosecond
Intioam_en	1	INT/OAM使能标识位, 0: 无效, 1: 有效; 仅有效时才需要解析ETS_Offset、Delay_Offset、QL_Offset、QP_Offset字段

如果INTIOAM_EN使能，TPU会根据上述指示完成以下操作。

表 4-39 TPU 操作

操作	操作方式
添加TM队列深度	从报文SOP位置开始，偏移QL_offset，填入TM发送指令中的6位队列深度信息。
添加TM队列占用率	从报文SOP位置开始，偏移QP_offset，填入TM发送指令中的8位队列占用率信息。
添加出口时戳	从报文SOP位置开始，偏移ETS_offset，截取RTC时间戳中的低2位秒信息，和低30bit纳秒信息，组合成32bit时戳信息{2' bs,30' bns}填入报文。
添加发送接收时间差	根据RTC当前时间戳，对ITS进行减计算得到时间差的绝对值，从报文SOP位置开始，偏移Delay_offset，将时间差的绝对值填入报文。



须知

1. 对于所有类型的发文，其偏移量计算均以报文开始部分（SOP）作为基准点。在计算偏移时，微码或者CPU需要考虑报文中位于DMAC之前的特殊字段，例如板间头、1588 OPCCDOE等。
2. 需要注意INT/IOAM头部信息不会被用作PPP查找关键字来查询流表和分发表。
3. 经过与相关微码开发人员确认，根据IOAM草案的规定，封装的最大长度为24个4Byte单位加上3个4Byte单位，总计27个4Byte单位。考虑到多级设备级联的情况，最多支持9层级联封装，即总长度可达243个4Byte单位（27*9）；然而，在当前协议未扩张域的情况下，实际应用场景中通常不会超过255个4Byte单位。因此，Hi309a接收描述符中的OL4LEN字段的最大位宽设定为8位。

4.4.11 多模隔离

在多模环境中，通过VFID实现页表隔离，最多支持5个VF。各个缓存池（Pool）之间是完全独立的。

对于所有报文，首先解析出其VQI（虚拟队列标识），然后根据VQI与Pool之间的映射关系确定对应的Pool信息。即使某个Pool中没有可用缓存，也不会影响其他Pool接收报文的能力。

每个Pool能够独立配置缓存申请启动状态、预申请缓冲区的FIFO深度、基于服务质量（QoS）的丢弃策略等参数。每个Pool的每个QoS级别都有独立的丢弃阈值。

- 单模复位功能：允许全局配置某些VF为disable状态。
 - 当VF被设为禁用时，从接收端来看，来自这些VF的报文会被直接丢弃。
 - 而在发送端，则不会向这些VF发送任何数据包。
- 资源回收机制：在系统升级或单模disable期间，可以关闭特定Pool的本地缓存申请，使得软件可以直接读取该Pool的Buffer内容。

物理功能（PF）能够访问所有寄存器，并且可以通过TOP_SR_IOV_CTL.vf_en_bitmap（偏移地址：0x0108）控制是否允许访问VF寄存器。每个VF可以独立启用或禁用，具体操作包括配置PPP内部寄存器PPP.PPE_PPP_VF_CNT_CFG（偏移地址：0x8B00）以选择需要统计的模式，以及通过PPE_PPP_VF_DIS_DROP_PKT_CNT（偏移地址：0x8B18）获取丢弃报文的统计数据；发送方向侧通过查询TPU内部寄存器PPE_TPU_TM_DROP_CMD_CNT（偏移地址：0x11028+tnl_num*0x40）来获取丢弃报文的统计信息。

- 中断报告：每个模式的中断事件都是独立的，并依据VFID进行上报。
- 访问DDR或加速器BMU、POE：每个VF需统一配置其streamid，格式为{asid[15:0],VFID[7:0]}。在访问总线时，根据VFID转换成对应的streamid。接收方向上，VFID由报文解析获得；发送方向上，VFID可以从CPU推送的描述符地址或TM推送的消息中获得。访问BMU时，使用的是逻辑Pool的VFID。

须知

当PF（物理功能）通过PF-VF或者特定VF（虚拟功能）尝试访问五个VF共享的物理资源时，若该资源尚未分配给请求的VF，则读取操作会返回该寄存器的上一次存储值。



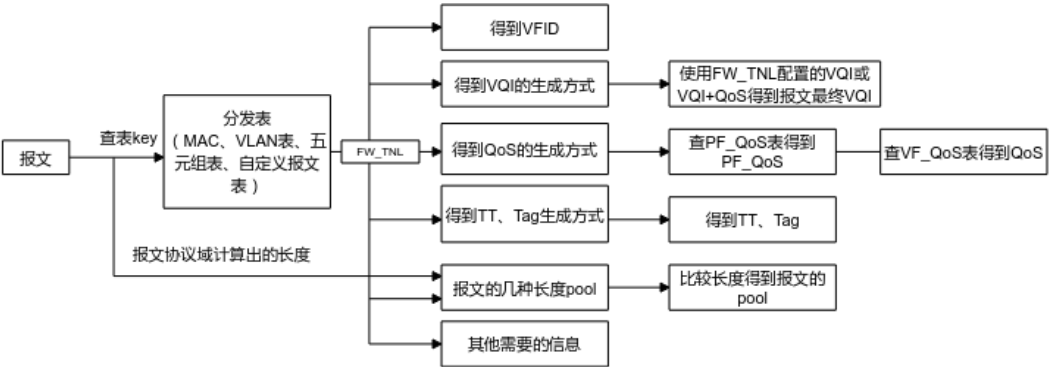
4.4.12 典型报文处理方式

典型的报文处理流程包括以下步骤：

- 步骤1** 报文首先解析以提取用于查表所需要的Key，例如MAC地址、VLAN ID、五元组、端口编号及报文长度等。
- 步骤2** 利用这些关键字段进行查找，确定报文对应的转发FW_TNL，共有16*16个转发隧道，每个子集包含16个隧道，每个子集可分配给一个VF。通过此查找过程，即可获得报文所属的VFID。
- 步骤3** 在选定的FW_TNL中，获取配置的信息，包括VQI（原Group的演化参数）、QoS、TT、Tag、Pool等。
- 步骤4** 根据不同的分发策略，综合上述信息，确定报文最终的分发结果，包括VQI、QoS、TT、Tag和Pool。
- 步骤5** 最后，利用指定的Pool将报文存储至内存，并将分发消息推送至POE。

----结束

图 4-65 典型报文的处理流程



4.4.12.1 Checksum 处理

本系统支持对包含隧道封装的L3/L4 层checksum的处理。具体包括了隧道外层IPv4头部和L4 checksum计算，以及隧道内部IPv4头部与L4协议数据单元的 checksum计算。如表4-40所示，列举出了各种不同类型的报文及其对应checksum的处理场景。

表 4-40 不同报文类型的 checksum 处理（OL3 for Outer L3，OL4 for Outer L4）

Packet Type	TX L3/OL3 checksum	TX L4/OL4 checksum	RX L3/OL3 checksum	RX L4/OL4 checksum
IPv4 + Unknown L4T	Y	N	Y	N
IPv6 + Unknown L4T	N	N	N	N
IPv4 + TCP/UDP/SCTP	Y	Y	Y	Y



Packet Type	TX L3/OL3 checksum	TX L4/OL4 checksum	RX L3/OL3 checksum	RX L4/OL4 checksum
IPv6 + TCP/UDP/ SCTP	N	Y	N	Y
VxLAN with IPv4	Y	N	Y	Y
NvGRE with IPv4	Y	N	Y	N
Geneve with IPv4	Y	N	Y	Y
CAPWAP with IPv4	Y	N	Y	Y
VxLAN with IPv6	N	N	N	Y
NvGRE with IPv6	N	N	N	N
Geneve with IPv6	N	N	N	Y
CAPWAP with IPv6	N	N	N	Y

- L3 层代表IP协议头部，包括IPv4和IPv6头部。
- L4 层则涉及传输层协议，如UDP/TCP/SCTP报文。
- L3 checksum仅仅适用于IPv4报文。
- L4 checksum则涵盖UDP/TCP的checksum及SCTP的CRC校验。

如表4-40所示，隧道报文仅列出了外层OL3和OL4的checksum。内层报文的checksum和处理方式与普通报文一致。在接收方向，硬件将非VxLAN、非NVGRE、非GENEVE或者非CAPWAP的隧道报文视为普通报文，仅处理外层的L3/L4 checksum。

4.4.12.1.1 发送侧 Checksum 计算

在软件中，可以通过设置发送描述符中的特定域来指示硬件是否需要进行checksum计算。硬件会根据描述符中的报文相关信息，遵循以下原则对报文进行checksum计算：

L3checksum计算：

- 当发送描述符中的L3_CS被设置为1，PPE（ Packet Processing Engine ）将执行L3层的checksum计算。
- PPE可以计算非隧道报文的IPv4 checksum以及隧道报文隧道内外层IPv4 checksum。

L4checksum计算：

- 若发送描述符中的L4_CS设置为1，则PPE将执行L4层的checksum计算。
- PPE发送侧PA（ Packet Acceleration ）会根据预设的模板识别报文类型，该模板与接收侧的识别标准一致。
- PPE可以计算非隧道报文UDP/TCP的checksum和SCTP CRC，以及隧道报文隧道内层UDP/TCP的checksum和SCTP CRC。

特殊隧道报文处理：



- 对于某些无法识别的特殊隧道报文，当其内部L4协议类型为UDP并且发送描述符中的隧道指示（TUN_TYPE）表明这是一条隧道报文时，PPE将会尝试计算其内部L4checksum。
- 在这种情况下，PPE会根据描述符中的OL2_LEN/OL3_LEN/OL4_LEN域段跳过隧道外层头（或多层隧道头），然后解析隧道内层并根据指示进行隧道内层的checksum计算。
- 请注意，使用此功能时，外层隧道头不能与发送侧PA隧道模板进行匹配。
- checksum计算时的报文头部长度限制：
L3 checksum：从IPv4 version 字段开始，计算长度为IPv4 IHL长度。最大60Byte。
L4 checksum：从UDP/TCP/SCTP报文头起始，计算结束位置取决于报文类型：
 - UDP：使用UDP length域所填的长度的低14位确定结束位置，PPE支持的最大长度报文均支持checksum计算。
 - TCP/SCTP：结束位置由IP 头中的total length域的低14位确定，同时，PPE支持的最大长度报文均支持checksum计算。

说明

1. IPv6头部没有checksum字段，因此无需进行checksum计算。
2. 对于隧道报文的L4 checksum，PPE仅支持计隧道内层的checksum；软件自行计算隧道内层和外层的checksum，或将外层UDP checksum填0。
3. 对于padding报文，如果可以通过IP头部长度信息识别出来，则PPE可以支持L4部分的checksum校验和生成，且padding的内容不会参与checksum计算；否则，硬件不支持类似checksum的生成。

4.4.12.1.2 接收端 Checksum 校验

- 接收端的checksum检查功能默认开启，并且可以由软件进行控制。
- 接收描述符中的L3L4P字段用于指示硬件是否执行了checksum校验，若硬件已进行校验，则将该位设置为0b1。
- 接收描述符中的OL3E和L3E分别代表隧道外层和内层数据包的L3 checksum校验结果。
- 接收描述符中的OL4E和L4E则分别对应隧道外层和内层数据包的L4 checksum校验结果。

说明

IPv6头部没有checksum字段，因此无需进行checksum计算；

IPv6报文头的最大长度为512Bytes；

如果某个端口绑定的功能支持GRO（Generic Receive Offload）功能，则该端口的checksum功能必须启用。

4.4.12.1.3 UDP Checksum 为 0 的处理

UDP checksum为0的时候，硬件会按照下[表4-41](#)进行处理。



表 4-41 UDP checksum 为 0 时的处理

L4e/EL4e	Tunnel Packet	-	-	-	non-Tunnel Packet	-
	Outer	-	Inner	-	-	-
	UDP = 0	UDP != 0	UDP = 0	UDP != 0	UDP = 0	UDP != 0
IPv4	Disable	Enable	Disable	Enable	Disable	Enable
IPv6	Disable	Enable	Error	Enable	Error	Enable

在表4-40中，不同报文类型的checksum处理规则如下：

- 对于UDP报文，这里的checksum指的是UDP头域中的UDP checksum字段。
- 当设置为Disable时，接收端（RX）不需要执行L4层 checksum校验。发送端（TX）的操作依据描述符指示进行，并且L4E/OL4E标志被标记为无效。
- 当设置为Enable时，接收端（RX）需要进行L4层checksum校验，发送端（TX）的操作同样遵循描述符指示，但在checksum校验出错的情况下，L4E/OL4E标志会被标记为有效。
- 当设置为Error时，则表示存在错误情况，此时L4E/OL4E标志会直接被标记为有效。

说明

IPv4 UDP checksum为0表示不进行checksum校验，发送和接收描述符的L4E/OL4E按照L4 checksum为正确填写；

非隧道或隧道内IPv6 UDP checksum为0表示非法报文，发送和接收描述符的L4E/OL4E按照L4 checksum为错误填写。

4.4.12.1.4 异常报文的 Checksum

在处理网络数据包时，可能会遇到以下几种异常情况：

- 对于IPv4报文，其IP头部长度应不小于20Byte。若发现IP头部长度小于20Byte，则视为异常。
- 对于IPv6报文，其IP头部长度应不小于40Byte。若发现IP头部长度小于40Byte，则视为异常。
- 对于UDP报文，其UDP头部长度应不小于8Byte。若发现UDP头部长度小于8Byte，则视为异常。
- 对于TCP报文，其TCP头部长度应不小于20Byte。若发现TCP头部长度小于20Byte，则视为异常。
- 对于SCTP报文，其SCTP头部长度不应小于16Byte。若发现SCTP头部长度小于16Byte，则视为异常。

L3L4P处理规则：

- 异常情况处理：
 - 如果出现任何异常情况，L3L4P应被设置为0。此时，描述符中的OL3E/L3E/OL4E/L4E/UDP0域均无意义。



- 无需checksum处理的情况：
 - 当前报文不需进行checksum处理，L3L4P应被设置为0，此时描述符中的OL3E/L3E/OL4E/L4E/UDP0 域均无意义。
- IPv6+UDP报文checksum为0的情况：
 - 对于IPv6+UDP报文，若其checksum等于0，则L3L4P应被设置为1，同时L4E置应被设置为1，而L3E应被设置为0。
- IPv6隧道报文内层UDPchecksum为0的处理：
 - 若IPv6隧道报文的内层UDP checksum等于0，L3L4P应被设置为1，L4E应被设置为1，OL3E/L3E/OL4E的状态则根据checksum的结果来决定。
- 接收隧道报文的特殊处理：
 - 对于接收的隧道报文，如果内层是IPv4+SCTP报文，且SCTP头长度小于16Byte，并且payload长度为0，L3L4P应被设置为0。
- 其他情况：

在其他情况下，L3L4的状态取决于checksum使能的有效性以及checksum的实际计算结果。

4.4.13 临终遗言报文

支持检测到dyinggasp输出信号有效后，停止发送正常报文，反复轮询发送指定报文给对接芯片。软件可以对帧Flag域、帧Code域、FCS域（硬件不会计算dyinggasp帧的FCS，需要软件计算后告诉给硬件）进行配置，并且需要在dyinggasp输出信号有效前配置完毕。dyinggasp报文具有最高发送优先级，可以截断正在发送的数据帧。dyinggasp的帧格式如表4-42所示。

表 4-42 dyinggasp 的帧格式

Ocater	Field	Value
6	Destination Address	0x01-80-c2-00-00-02
6	Source Address	ONU/LLIDO MAC Address
2	Length/Type	88-09 [Slow Protocols]
1	Subtype	0x03 [OAM]
2	Flags	16'h0002
1	Code	8'h00
42	Data/Pad	8'h00
4	FCS	FCS

操作流程如下：

- 软件需预先配置帧的Flag域和帧Code域；FCS域不需要手动配置，当MAG构造好报文后，MAC层会自动添加FCS域。
- 当检测到dyinggasp信号时，MAG应自行停止发送侧的pause响应。



- 然后，MAG应启动周期性发送dyinggasp报文的过程。

须知

请注意，系统在检测到即将断电时会自动发送临终遗言报文给对接设备。此功能为单向通知，一旦被触发并开启后，必须复位PPE才能恢复数据传输，否则数据流将无法自动恢复。

4.5 编程接口

4.5.1 数据结构

表 4-43 PPE ADD_PACKET 数据结构 0x41（Order_TagValue 与 Affinity_TagValue 复用场景）

3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0		
Lo ca tio n		Vq i_R sv		Vqi[9:0]										SourcePort[7:0]										T_Len[3 :0] =4'b010 0		T_TagV alue[3:0] =4'b000 1		M sg _L en =2 'b 01					
R s v	Order_Rsv			Or de r_ Ty pe		Order_TagValue[23:0]/Affinity_TagValue[23:0]																											
Addr[31:0]																																	
Re l	R l s _ l i n d	Re se r ve d	Addr _Rsv		Pref_Of fset1		Pref_Of fset2		Addr[47:32]																								
Reserved																																	
Reserved																																	



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
Hac_Occ_Timestamp_Value																					T_Len[3:0] =4'b0001			T_TagV alue[3:0]]=4'b0011 or 4'b0111 or 4'b1000 (Timest amp)							
Hac_Occ_CRC							Hac_Oc c_Add_I D			-	Hac_Occ_Ost d_SN					Hac_ Occ_ DFX_ VFID			Hac_ Occ_ FRD_ Qos			T_Len[3:0] =4'b0001			T_TagV alue[3:0]]=4'b0100 (Hardw are)						
说明																															
POE基础字段							SEC输入包任务描述符、SEC输出包任务描述符																								
Hardware TLV填写说明							SEC输入包任务描述符、SEC输出包任务描述符																								
{Rls_Ind, Rel}填写说明							同一个报文多个分片必须都释放回BMU，或者都不释放回BMU。 如果这个报文的buffer不是PPE申请的BMU的{Rls_Ind,Rel} 分别填写{1'b0, 2'b11}。 如果这个报文的buffer是PPE申请的BMU的：单buffer报文：头包分离时填写{1'b1,2'b01}；非头包分离，填{1'b1,2'b00}; 多buffer报文：头包分离/非头包分离，填{1'b1, 2'b01}。																								

表 4-44 ADD_SPE_WORK，PPE 上送 1588 时戳到 NP 的消息数据结构

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
Lo ca tio n	Vq i_R sv	Vqi[9:0]											SourcePort[7:0]					T_Len[3:0] =7	T_TagV alue[3:0]]=4'b1000	Ms g_ Le n =2'b01										
CMD_type[7:0] (固定为254)								Rsv																						

3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Timestamp[31:0]																															
Timestamp[63:32]																															
Timestamp[95:64]																															
Timestamp[127:96]																															
Rsv																															
Hac_Occ_CRC				Hac_Occ_Add_ID				Hac_Occ_Os td_SN				Hac_Occ_DFX_VFID				Hac_Occ_FRD_Qos				T_Len[3:0] =4'b0001				T_TagV alue[3:0]]=4'b0100 (Hardw are)							
说明																															
CMD_type				254:PPE上送1588时戳消息。																											
Timestamp				1588时戳值，信息格式为{35'h0，port_id[4:0]，pid[7:0]，seqinfo[15:0]，秒[31:0]，原时戳[31:0]（秒[31:30]，纳秒[29:0]）}																											

表 4-45 发送描述符首片节点数据结构

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0												
PrevAddr[31:0]																																	
PrevAddr_Reserved																PrevAddr[47:32]																	
NextAddr[31:0]																																	
NextAddr_Reserved																NextAddr[47:32]																	
RefItselfAddr[31:0]																																	
RefItselfAddr_Reserved																RefItselfAddr[47:32]																	
DataOffset																DataLength																	
ChainOffset																T_Len=6'b001000						T_TagValue=6'b000001						Dsc_Len=4'b0001					
PktLen																		MTU_in dex						Status				Own					



3 1 0	3 0 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
L5_ID			L4_ID			L3_ID			VL A N	D M A C	i n t l o a m - e n	W B C	Pool_num				E N C A P	spl it _ ty p e	i p 4 _ o p t _ t y p e	r e s e r v e d	L 4 _ C S	C N _ T a g E n								
PktPseudoHdrPtr[31:24]						PktPseudoHdrPtr[23:16]/L4LEN						PktPseudoHdrPtr[15:8]/L3LEN						PktPseudoHdrPtr[7:0]/L2LEN												
FlowId												PktPseudoHdrPtr[47:32]																		
ITS																														
QL_Offset						QP_Offset						ETS_Offset						Delay_Offset												
SC_target						cacheAttr						SC	X S E C - E N C - P R O F I L E	XSEC_ENC_SC				P A I B D - I N D	PT P M	Y V L D 2	Y V L D 1	A S T S	A S T P							
STS/YID2						YID1						STSoffset						STPoffset												
-																														
字段			位说明 宽																											
Dsc_Len			4	指示变长报文描述符长度，单位512bit，（ 0:256bit， 1:512bit ）。																										
T_TagValue			6	数据结构按照TLV方式定义，类型标识T_TagValue。																										
T_Len			6	数据结构按照TLV方式定义，结构长度，单位32bit。																										
PrevAddr			64	对于链表方式存储的报文，该域表示上一段链表节点结构所在地址（按Cache-Line 64Byte对齐）。																										



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0	
					【注意】该部分为软件或其他加速器预留，PPE只实现单向链表。																											
NextAddr					6 4	对于链表方式存储的报文，该域表示下一段链表节点结构所在地址（按Cache-Line 64Byte对齐）。																										
RefltselfAddr					6 4	该分片报文数据BUFF块基地址，该地址+DataOffset是报文数据起始位置，0表示首节点无数据，只允许首节点不带报文；（按Cache-Line 64Byte对齐）。																										
DataLength					1 6	表示该buffer报文数据的长度，单位：字节。																										
DataOffset					1 6	报文数据首地址相对于RefltselfAddr的偏移，单位：字节。																										
ChainOffset					1 6	该片报文数据相对于整报文的偏移，单位：字节。（该域为软件预留部分）																										
Own					4	软件填写0xA提交任务，PPE硬件处理完成后回写0x5;BMU解析处理完成后回写0x6；PPE是否检查own域，基于物理通道可配。																										
Status					4	硬件发送完成状态异常类型，0：正常；其他：异常类型 0x0: OK for PPE transfer 0x1: error for PPE transfer 0x2: desc error 0x3: fatal error other:reserved																										
MTU idx					4	309a该域无意义。																										
PktLen					2 0	报文总长度，单位：字节数，0是非法值，软件需要保证pktlen=所有片的send_size总和，否则硬件会作为错包处理。如果需要硬件封装报文头，表示payload长度，MF报文最大支持1MB-1，其余报文最大支持64KB。如果不需要硬件封装报文头表示报文总长。																										
CN_TagEn					1	309a该域无意义。																										
L4_CS					1	报文发送时，控制非IP分片报文是否对四层计算Checksum： 0：没有使能：如果是硬件封装的报文，硬件固定写0；如果是软件封装的报文，硬件不处理，维持原来的值；1：使能。																										
L4_CS_P					1	309a该域无意义。																										



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
SC		2	<p>Switch control: (如果要启用SC, 需将PPE TNL配置为LSW模式)。</p> <p>00: No Switch control. The packet is routed according to the forwarding table. 关闭SC;</p> <p>01: Uplink packet. The packet is transmitted to the network bypassing the forwarding table . (SC打开, bypass LSW查表, 发送到指定的网口号);</p> <p>10: Local packet. The packet is transmitted only to local VSI's according to the forwarding table. (SC打开, LSW查表, 根据查表结果进行定向, 把去往MAC口的滤掉);</p> <p>11: Target VSI. The packet is transmitted to a specific VSI defined in the TARGET_VSI, the target VSI and source VSI must belong to the same MAC port. (SC打开, bypass LSW查表, 发送到指定的PPE TNL)</p>																												
ip4_opt_type		2	309a该域无意义。																												
split_type		2	309a该域无意义。																												
ENCAP		1	309a该域无意义。																												
Pool_Num		1	发送缓存本地释放的Pool号 (vf的pool号), 配合LC使用, 该报文不管是链式或描述符报文分离, 所有需要释放的buffer必须是同一个pool, 否则会出现缓存踩踏。发送任务描述符中 (Rls_Ind, Rel) 所指示的需要释放Buffer地址都属于同一个POOL (即不支持同时释放分离存放的描述符和报文)。																												
LC		1	发送报文缓存本地释放指示, (当发送需要释放缓存时, 如果该bit为1, 缓存优先释放到PPE内部), Pool_Num需要填充正确:																												
			0b0: 当发送需要释放缓存时, 直接释放给BMU;																												
			0b1: 当发送需要释放缓存时, 优先释放到PPE内部。																												
WB		1	是否回写状态域status和own域 0: 不回写 1: 回写。																												
L2_ID, L3_ID, L4_ID, L5_ID		6	PKtType。																												
SC_target		8	当SC=2'b11时, bit[4:0]表示去往的tnl号, 其余位填0; 当SC=2'b01时, bit[4:0]表示去往的MAC口, bit[11:5]固定填0。																												
PktPseudo HdrPtr		8	309a该域无意义。																												
FlowId		6	309a该域无意义。																												



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
L2LEN		8	二层头长度，包括mac地址、VLAN、CN-tag、SNAP、ETHERTYPE等，即 IP头在首分片的偏移位置（Mac头的长度），单位：2Byte（有效值需要<=64，相当于L2LEN最大值是128字节）；仅ENCAP为0时有效。																												
L3LEN		8	IP头的长度，单位：4Byte，对于IPV4有效取值5~15；仅ENCAP为0时有效。包括IPv4 option长度，但是不包括IPv6的AH/ESP/分片扩展头长度。																												
L4LEN		8	L4报文头长度，以4Bytes为单位。																												
TimeStam p_NanoSe cond		3 0	309a该域无意义。																												
T2VLD		1	309a该域无意义。																												
T2M		1	309a该域无意义。																												
TimeStam p_Second		3 2	309a该域无意义。																												
ASTP		1	该报文是否要添加时间戳使能（用于Y.1731）。0：禁止；1：使能。																												
STPoffset		8	ASTP有效时，添加时间戳距报文头的偏移值。单位字节数。																												
ASTS		1	该报文是否要添加统计信息（用于Y.1731）。0：禁止；1：使能。																												
STSoffset		8	ASTP有效时，添加统计信息距报文头的偏移值。单位字节数。																												
STS/YID2		8	ASTS有效时，该报文的流ID，将该条流的发送报文统计值添加到报文里。YVLD2有效时，需要参与Y.1731统计流2的ID。仅支持32条流，低5bit有效。																												
YVLD1		1	该报文是否需要参与Y.1731的流1统计，0：禁止；1：使能。																												
YID1		8	YVLD1有效时，需要参与Y.1731统计流1的ID。仅支持32条流，低5bit有效。																												
YVLD2		1	该报文是否需要参与Y.1731的流2统计，0：禁止；1：使能。																												
PTPM		2	bit[0]0：不需要硬件进行打戳或修改修正域的1588报文；1：需要硬件进行打戳或修改修正域的1588报文。bit[1]保留域，硬件不关注，软件填写为0。																												
BI		1	保留域，硬件不关注，软件填写为0。																												
cacheAttr		8	cache属性 [0]CLEANINVALID [1]Force-Allocate。强制cache分配其他：保留。																												
PA_BD_IN D		1	1有效表示TXPA使用描述符指示的L2/L3 LEN进行报文解析。																												



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
XSEC_ENC_SC					5	xsec_enc_sc: security channel。																								
XSEC_ENC_PROFILE					1	0: 不加密; 1: 加密。																								
ITS					3	Ingress Timestamp, 入口时戳, 2bit Second + 30 bits Nanosecond。																								
					2																									
Delay_Offset					8	Transit Delay Offset, 时延偏移, 1 bit overflow指示位 + 31 bit 时延 (ns), 编辑4Byte, 如果超2^31-1, 则0x80000000; 0则不编辑。																								
ETS_Offset					8	Egress Timestamp Offset, 出口时戳偏移, 编辑4Byte; 0则不编辑; 出口时戳, 2bit Second + 30 bits Nanosecond。																								
QP_Offset					8	Queue Occupancy Percentage Offset, 队列占用率, 编辑4Byte; 0则不编辑。																								
QL_Offset					8	Queue Length Offset, 队列深度, 编辑4Byte; 0则不编辑。																								
Intloam_en					1	INT/IOAM使能标识位, 0: 无效, 1: 有效; 仅有效时才需要解析 ETS_Offset、Delay_Offset、QL_Offset和QP_Offset字段。																								

表 4-46 发送/接收描述符非首片节点数据结构

3	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0								
PrevAddr[31:0]																													
PrevAddr_Reserved															PrevAddr[47:32]														
NextAddr[31:0]																													
NextAddr_Reserved															NextAddr[47:32]														
RefltselfAddr[31:0]																													
RefltselfAddr_Reserved															RefltselfAddr[47:32]														
DataOffset															DataLength														
ChainOffset															T_Len=6'b000000					T_TagValue=6'b0000000					Dsc_Len=4'b00000				
-																													
字段		位说明																											
Dsc_Len		4 指示变长报文描述符长度，单位512bit，（ 0:256bit， 1:512bit ）。																											



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
T_TagValue		6	数据结构按照TLV方式定义，类型标识T_TagValue。																												
T_Len		6	数据结构按照TLV方式定义，结构长度，单位32bit。																												
PrevAddr		6	对于链表方式存储的报文，该域表示上一段链表节点结构所在地址																												
		4	(按Cache-Line对齐)。 注意：该部分为软件或其他加速器预留，PPE只实现单向链表。																												
NextAddr		6	对于链表方式存储的报文，该域表示下一段链表节点结构所在地址																												
RefItselfAddr		6	该分片报文数据BUFF块基地址，该地址+DataOffset是报文数据起始位置。																												
DataLength		1	表示该buffer报文数据的长度，单位：字节。																												
DataOffset		1	报文数据首地址相对于RefItselfAddr的偏移，单位：字节。																												
ChainOffset		1	该片报文数据相对于整报文的偏移，单位：字节。（该域为软件预留部分）																												

表 4-47 接收描述符首片节点数据结构

3	3	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0												
PrevAddr[31:0]																																	
PrevAddr_Reserved																PrevAddr[47:32]																	
NextAddr[31:0]																																	
NextAddr_Reserved																NextAddr[47:32]																	
RefItselfAddr[31:0]																																	
RefItselfAddr_Reserved																RefItselfAddr[47:32]																	
DataOffset																DataLength																	
ChainOffset																T_Len=6'b001000						T_TagValue=6'b000010						Dsc_Len=4'b01					
PktLen																				Status						Own							

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
L5_ID				L4_ID				L3_ID				VLAN		DMAC		XSB		WLC4P		Pool_num				udp0		split_type		mpls_num		OVLAN		L_PBK	
L3LEN								L2LEN												TUN_TYPE				L4LEN									
VLAN TAG														outer VLAN TAG																			
TimeStamp_NanoSecond																																	
TimeStamp_Second																																	
YID1_STS																																	
YID2_STS																																	
-																																	
字段		位说明																															
Dsc_Len		4 指示变长报文描述符长度，单位512bit，（ 0:256bit， 1:512bit ）。																															
T_TagValue		6 数据结构按照TLV方式定义，类型标识T_TagValue。																															
T_Len		6 数据结构按照TLV方式定义，结构长度，单位32bit。																															
PrevAddr		6 4 硬件忽略，软件可使用作为前向指针。																															
NextAddr		6 4 下一个分片数据描述指针，最后分片填写NULL。																															
RefltselAddr		6 4 该分片报文数据BUFF块基地址，即报文首描述符的起始地址。																															
DataOffset		1 6 报文数据首地址相对于RefltselAddr的偏移，单位：字节。																															
DataLength		1 6 表示该buffer报文数据的长度，单位：字节。																															
ChainOffset		1 6 该片报文数据相对于整报文的偏移，单位：字节。（该域为软件预留部分，硬件固定填0）。																															
Own		4 硬件处理固定写0x05。																															



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
Status		8	报文接收状态异常类型：具体参考PPE FS《1.4.5.4 畸形报文的检查》（注：PA对于报文错误的处理方式：L2错误的话，L3、L4错误指示固定为0；L3错误，L4错误指示固定为0）bit[0]：为1表示L2错误；bit[1]：为1表示L3错误，对于CAPWAP报文既包含外层也包含内层；bit[2]：为1表示L4错误，对于CAPWAP报文既包含外层也包含内层；bit[3]：为1表示报文有drop，ssu内部mem存不下出现截断；bit[4]：为1表示单buf不够存下完整报文，在链式申请时出现如下情况导致报文发生截断：MAM返回drop、disable、empty且重复申请达到最大次数并配置为丢弃；bit[5]：reserved；bit[6]：reserved；bit[7]：为1表示做了报文在处理过程中出现了ecc 2bit错误。																												
PktLen		2	报文的总长度，单位：字节，有效范围：1~65536。【19:17】固定为0。																												
mpls_num		2	mpls层数。																												
split_type		2	IP分片模式 0：不分片 1：IP分片； 2/3：保留。																												
Pool_Num		5	接收报文使用缓存的Pool号（vf的pool号，固定填写的是报文数据所用buffer的pool号）。																												
WB		1	是否回写状态域status和own域（PPE接收时固定写0）。 0：不回写； 1：回写。																												
TUN_TYPE		4	隧道类型指示。 0x0：非隧道报文； 0x1：MAC in UDP 隧道报文（1230无意义）； 0x2：NVGRE隧道报文（1230无意义）； 0x3：CAPWAP控制明文报文，隧道内层key不可信； 0x4：CAPWAP数据Keep-Alive明文报文，隧道内层key不可信； 0x5：CAPWPA数据data payload明文报文； 0x6：CAPWAP密文，隧道内层key不可信； 0x7~0xE：保留； 0xF：未知报文或基带自定义RAW报文。																												
LPBK		1	Loopback指示，表示该报文是从host发出的经过环回到接收侧。 注意：在DMAC域段为镜像，提取的信息不可用。																												
L3L4P		1	当软件使能L3或者L4 checksum校验的时候，并且接收的报文为IPv4/IPv6/TCP/UDP/SCTP类型，该域置为1表示硬件已经进行了L3L4 checksum处理。																												
udp0		1	接收的报文为UDP报文并且其checksum等于0的时候该域置位1。																												
XSEC_EN		1	接收侧，0：代表报文原来是明文，Xsec没有进行解密操作；1：代表报文原来是密文文，Xsec有进行解密操。																												



3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	1 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
VLAN, DMAC, L3_ID, L4_ID, L5_ID		1 6	PKtType, 注意：对于CAPWAP报文，该域描述的是隧道内层报文。																												
L2LEN		1 2	非CAPWAP报文：二层头长度，包括mac地址、VLAN、CN-tag、SNAP、ETHERTYPE等，即 IP头在首分片的偏移位置（Mac头的长度），单位：2Byte； CAPWAP报文：报文外层SMAC开始到隧道内2层头结束的长度。																												
L3LEN		8	IP头的长度，单位：4Byte，对于IPV4有效取值5~15；包括IPv4 option长度，但是不包括IPv6的AH/ESP/分片扩展头长度。注意：对于CAPWAP报文，该域针对的是隧道内层报文																												
L4LEN		8	4层头的长度，单位：4Byte 注意：对于CAPWAP报文，该域针对的是隧道内层报文注意：对于CAPWAP报文，该域针对的是隧道内层报文。																												
OVLAN_T AG		1 6	非隧道报文或者隧道外层报文的外层VLAN tag： 指在LSW出口剥离之前的vlan tag，如果无vlan，则为配置的VLAN。 如果接收的是隧道报文，该域表示的是靠近隧道外SMAC的tag。 如果接收的是非隧道报文，该域表示的是靠近SMAC的tag。 注意：在DMAC域段为镜像，或者Cport为DSA口时，提取的信息不可用。																												
VLAN_TA G		1 6	非隧道报文或者隧道外层报文的内层VLAN tag： 指在LSW出口剥离之前的vlan tag，如果无vlan，则为配置的vlan。 在隧道报文时，指隧道外层报文靠近L3头的tag，非隧道报文时指的是靠近L3头的tag。 注意：在DMAC域段为镜像，或者Cport为DSA口时，提取的信息不可用。																												
TimeStam p_NanoSe cond		3 2	报文时间戳，纳秒级时间；对于来自MAC的报文，高2bit填写SSU送过来的2bit s值；对于来自CPU的报文WORD12填写RTC送进来的timestmap_ns。																												
TimeStam p_Second		3 2	报文时间戳，秒级时间。																												
YID1_STS		3 2	309a该域无意义。																												
YID2_STS		3 2	309a该域无意义。																												



4.6 应用配置

4.6.1 报文分发功能

步骤1 在PF上配置MAC地址、VLAN和五元组表，并为每个表项配置对应的FW_TNL。

步骤2 基于16个FW_TNL组，在PF上分别为每个FW_TNL配置对应的VFID。

步骤3 在PF上配置PF_QoS表。

步骤4 VF在每个对应的FW_TNL上配置其VQI、QoS、TT、Tag、Pool的生成方式。

步骤5 VF配置VF_QoS表。

----结束

4.6.2 测试网口功能

4.6.2.1 接收方向

步骤1 配置MAC，具体参考《Hi309a 寄存器手册》中MAC初始化流程。

步骤2 初始化、配置PPE，具体参考章节“[初始化](#)”。

步骤3 配置调试网口使能寄存器RPU.RPU_TEST_PORT_CFG（0x0BC0）使能相应的通道。

步骤4 配置超时寄存器RPU.RPU_TEST_PORT_TIMEOUT（0x0520+tnl_num*0x40）、接收报文数量寄存器RPU_TEST_PKT_CNT_FG（0x0524+tnl_num*0x40）。

步骤5 当PPE收到软件配置的报文个数后上报中断，或收到报文后没达到配置报文个数发生超时后也会触发中断，软件收到中断按照软件记录的Buffer地址直接去处理报文。

----结束

4.6.2.2 发送方向

步骤1 配置MAC，具体参考《Hi309a 寄存器手册》中MAC初始化流程。

步骤2 初始化、配置PPE，具体参考章节“[初始化](#)”。

步骤3 发送方向首先配置调试网口功能使能和调试网口通道号PPE_TPU_TEST_PORT_CFG（0x6000），配置调试网口总线配置PPE_TPU_TEST_PORT_USER_STRMID（0x6010）、PPE_TPU_TEST_PORT_SMMU_USER（0x6014）、PPE_TPU_TEST_PORT_SMMU_USER_SSET（0x6018）。

步骤4 CPU 通过推包寄存器PPE_CORE_ADD_TX_DB_WORD0（0x11E00+0x10*core_num）、PPE_CORE_ADD_TX_BD_WORD1（0x11E04+0x10*core_num）直接将发送描述符地址推送到PPE。

步骤5 PPE完成发送后将OWN域修改为0x5，软件查询到为0x5后表示报文已经发送完，可以回收Buffer。

----结束



4.6.3 各个表项配置流程

PPP的大部分表项和端口属性配置都采用间接访问机制，CHRW类型的寄存器为表项的数据，以直通表为例。

4.6.3.1 直通表

直通表写操作：

- 步骤1** 配置PPP_DIR_TBL_DATA，CHRW类型，写入配置值。
- 步骤2** 配置PPP_DIR_TBL_CFG.cfg_dir_tbl_sel和cfg_dir_tbl_addr，写入表项地址。
- 步骤3** 启动写操作，PPP_DIR_TBL_CFG.cfg_dir_tbl_cmd_en，先写0，再写1。
- 步骤4** 轮询PPP_DIR_TBL_CFG.cfg_dir_tbl_status，如果为0，表示操作完成。

----结束

直通表读操作：

- 步骤1** 配置PPP_DIR_TBL_CFG.cfg_dir_tbl_sel和cfg_dir_tbl_addr，写入表项地址。
- 步骤2** 配置PPP_DIR_TBL_CFG.cfg_dir_tbl_cmd为1。
- 步骤3** 启动写操作，PPP_DIR_TBL_CFG.cfg_dir_tbl_cmd_en，先写0，再写1。
- 步骤4** 轮询PPP_DIR_TBL_CFG.cfg_dir_tbl_status，如果为0，表示操作完成。
- 步骤5** 从PPP_DIR_TBL_DATA读取数据。

----结束

4.6.3.2 TCAM 配置

TCAM的配置分为X和Y，AD，在添加时，建议先写AD，再配置Y，最后配置X和vld，在删除时，先写X和vld。

以TCAM配置X和vld为例：

TCAM写操作：

- 步骤1** 先往PPP_FD_KEY_DATA_0~MSB，CHRW类型，写入数据。
- 步骤2** 选择表项和写入地址，PPP_FD_KEY_CFG.cfg_fd_tab_sel和cfg_fd_key_addr。
- 步骤3** 选择写入X，PPP_FD_KEY_CFG.cfg_fd_key_x为1，和相关的表项有效信号，PPP_FD_KEY_DATA_MSB.cfg_fd_key_x_vld为1。
- 步骤4** 选择写入的是key数据，PPP_FD_KEY_CFG.cfg_fd_key_ad_sel为0。
- 步骤5** 启动TCAM的写，PPP_FD_KEY_CFG.cfg_fd_key_cmd_en先写0，再写1。
- 步骤6** 轮询PPP_FD_KEY_CFG.cfg_fd_key_status，如果为0，表示操作完成。

----结束

TCAM读操作：

- 步骤1** 选择表项和写入地址，PPP_FD_KEY_CFG.cfg_fd_tab_sel和cfg_fd_key_addr。



- 步骤2** 选择写入X, PPP_FD_KEY_CFG.cfg_fd_key_x为1。
- 步骤3** 选择是key数据, PPP_FD_KEY_CFG.cfg_fd_key_ad_sel为0。
- 步骤4** PPP_FD_KEY_CFG.cfg_fd_key_cmd配置为0x1。
- 步骤5** 启动TCAM的读, PPP_FD_KEY_CFG.cfg_fd_key_cmd_en先写0, 再写1。
- 步骤6** 轮询PPP_FD_KEY_CFG.cfg_fd_key_status, 如果为0, 表示操作完成。
- 步骤7** 从PPP_FD_KEY_DATA_0~MSB读取相应数据, 查看PPP_FD_KEY_DATA_MSB.cfg_fd_key_x_vld是否1。

----结束

TCAM查找操作:

- 步骤1** 向PPP_FD_KEY_DATA_0~MSB, CHRW类型, 写入待查找的key。
- 步骤2** 配置PPP_FD_KEY_CFG.cfg_fd_tab_sel, 选择查找的表项(入口流表、出口流表还是分发流表); 配置PPP_FD_KEY_CFG.cfg_fd_key_ad_sel为0, 表示查找Key。
- 步骤3** 配置PPP_FD_KEY_CFG.cfg_fd_key_cmd为查找操作。
- 步骤4** 启动TCAM的查找, PPP_FD_KEY_CFG.cfg_fd_key_cmd_en先写0, 再写1。
- 步骤5** 轮询PPP_FD_KEY_CFG.cfg_fd_key_status, 如果为0, 表示操作完成。
- 步骤6** 操作完成后, 查看PPP_FD_STATUS是否命中表项以及命中表项的索引。

----结束

4.6.3.3 L2 表配置

L2表采用Hash实现方式, 命令丰富, 支持多种操作命令。

另外, 如果对端口学习限制和表项占用计数器等有修改操作需求, 是要伴随L2表的读操作一起的。

L2 表添加 key

- 步骤1** 配置好Key和关联数据AD, CHRW类型。

PPP_L2_MAC_LSB	PPP L2表 mac地址	reg	0x0900
PPP_L2_MAC_MSB	PPP L2表 mac地址和VLAN	reg	0x0904
PPP_L2_AD	PPP L2表关联数据	reg	0x0910

- 步骤2** 配置命令类型。
PPP_L2_CFG.cfg_l2_cmd为0。
- 步骤3** 配置命令使能。
PPP_L2_CFG.cfg_l2_cmd_en配置为0, 再配置为1。
- 步骤4** 查询状态。



PPP_L2_OP_STS.l2_op_status为1表示命令还没完成，继续查询；如果为0则表示命令已完成。查看ppp_l2_key_staus状态。

----结束

L2 表删除 key

- 步骤1 配置好Key，参照L2添加key。
- 步骤2 配置命令类型，参照L2添加key ， ppp_cfg_l2_cmd为2。
- 步骤3 配置命令使能，参照L2添加key。
- 步骤4 查询状态，参照L2添加key。

----结束

L2 表查找

- 步骤1 配置好Key，参照L2添加key。
 - 步骤2 配置命令类型，参照L2添加key ， ppp_cfg_l2_cmd为3。
 - 步骤3 配置命令使能，参照L2添加key。
 - 步骤4 查询状态，参照L2添加key。
 - 步骤5 如果命中，读取AD返回结果和命中位置。
- PPP_L2_OP_STS. l2_lookup_hit_position/ l2_lookup_hit_addr

----结束

L2 表按地址读

此命令可以按地址方式，直接把表项的内容和关联数据读出。

- 步骤1 配置好访问地址和访问对象。

PPP_L2_CFG	PPP L2表 操作命令	reg	0x0920	32		RW	0x00000000	reserved
								cfg_l2_addr_sel
								cfg_l2_addr

- 步骤2 配置命令类型，参照L2添加key ， ppp_cfg_l2_cmd为4。
- 步骤3 配置命令使能，参照L2添加key。
- 步骤4 查询状态，参照L2添加key。
- 步骤5 读取返回key和AD。

PPP_L2_MAC_LSB	PPP L2表 mac地址	reg	0x0900
PPP_L2_MAC_MSB	PPP L2表 mac地址和VLAN	reg	0x0904
PPP_L2_AD	PPP L2表关联数据	reg	0x0910

----结束

L2 表按地址写命令

此方式只作为DFX手段使用，软件在写Hash桶表项时，需要计算当前的key的hash值，保证在当前配置的hash地址内，否则，硬件查表时会查找不到。

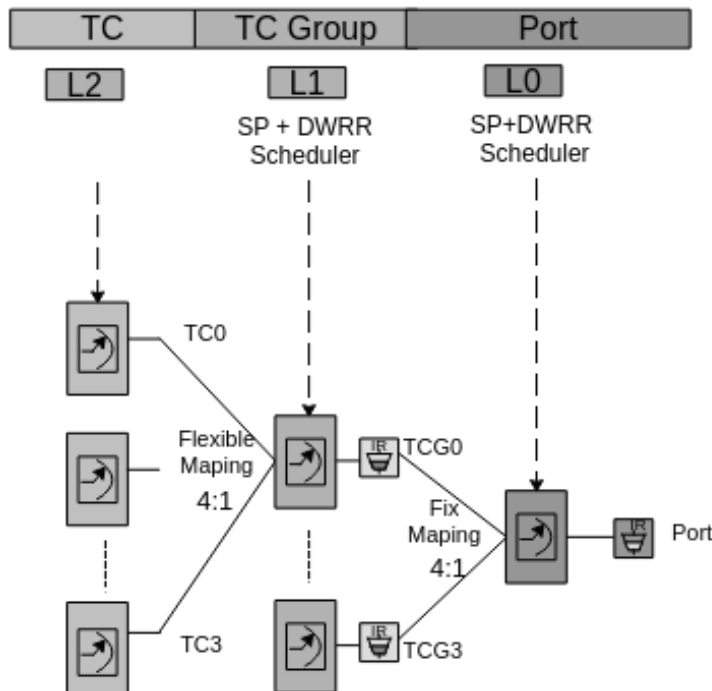
- 步骤1 配置好Key和关联数据AD，参照L2添加key。
- 步骤2 配置好访问地址和访问对象，参照L2按地址读。
- 步骤3 配置命令类型，参照L2添加key， ppp_cfg_l2_cmd为5。
- 步骤4 配置命令使能，参照L2添加key。
- 步骤5 查询状态，参照L2添加key。

----结束

4.6.4 ETS 配置

4.6.4.1 调度结构图

图 4-66 调度结构图





4.6.4.2 配置寄存器

表 4-48 配置寄存器

Reg Name	Description	*Offset Addr	*Width	Attribute	Field Name	Field Range	*Field Attribute	Field Default Value	Field Comments
ETS_TC_WEIGHT_0	ETS priority nodes weight configuration table (pri0~pri3)	0x43C0+port_num*0x4	32	RW	mac_tc_weight_cfg0	31:0	RW	0x01010101	bit[31:24]: 对应pri3。 bit[23:16]: 对应pri2。 bit[15:8]: 对应pri1。 bit[7:0]: 对应pri0。 mac weight based on priority。 当配置为0时，则认为是SP调度模式，否则为DWRR对应调度比例值。
ETS_TC_WEIGHT_1	ETS priority nodes weight configuration table (pri4~pri7)	0x4400+port_num*0x4	32	RW	mac_tc_weight_cfg1	31:0	RW	0x01010101	bit[31:24]: 对应pri7。 bit[23:16]: 对应pri6。 bit[15:8]: 对应pri5。 bit[7:0]: 对应pri4。 mac weight based on priority。



Reg Name	Description	*Offset Addr	*Width	Attribute	Field Name	Field Range	*Field Attribute	Field Default Value	Field Comments
ETS_TC G_WEIGHT	ETS priority Group nodes weight配置表	0x4440+ port_num*0x4	32	RW	mac_tcg_weight_cfg	31:0	RW	0x01010101	bit[31:24]: 对应pri3 bit[23:16]: 对应pri2。 bit[15:8]: 对应pri1。 bit[7:0]: 对应pri0。 mac weight based on TC。 当配置为0时，则认为是SP调度模式，否则为DWRR对应调度比例值。
ETS_TC G_SHA PING_MAC	MAC ETS priority Group nodes 级的 shaper 配置参数	0x4480+ port_num*0x10+ tcg_num*0x4	32	RW	reserved	31:26	RO	0x00	保留。



Reg Name	Description	*Offset Addr	*Width	Attribute	Field Name	Field Range	*Field Attribute	Field Default Value	Field Comments
ETS_TC G_MAPPING	ETS priority Group Mapping bitmap配置	0x4570+ port_num*0x4	32	RW	mac_tc cg_mapping_cfg	31:0	RW	0x000000FF	bit[31:24]: 对应TC3。 bit[23:16]: 对应TC2。 bit[15:8]: 对应TC1。 bit[7:0]: 对应TC0。 每一个bit表示一个对应priority节点: 0: 表示和对应TC不挂接; 1: 表示和对应TC挂接。
ETS_PORT_SHAPING	MAC ETS PORT node 级的 shaper 配置参数	0x45B0+ port_num*0x4	32	RW	reserved	31:26	RO	0x00	保留。
-	-	-	-	-	mac_port_shaper_cfg	25:0	RW	0x2851096	{ir_bs_s[4:0], ir_bs_b[4:0], ir_s[3:0], ir_u[3:0], ir_b[7:0]}:
ETS_WEIGHT_OFFSET	ETS 调度权重偏移值	0x45F0+ port_num*0x4	32	RW	reserved	31:4	RO	0x00000000	保留。



Reg Name	Description	*Offset Addr	*Width	Attribute	Field Name	Field Range	*Field Attribute	Field Default Value	Field Comments
-	-	-	-	-	mac_weight_offset_cfg	3:0	RW	0xE	mac权重粒度，内部真实使用的 credit = weight << weight_offset，配置范围 0~14。

4.6.4.3 链接关系配置举例

```
pri0, pri 1 ->TC0, tcg_maping_cfg[7:0] = 0x03;  
pri2, pri7 ->TC1, tcg_maping_cfg[15:8] =0x84;  
pri3, pri4, pri 5 ->TC2, tcg_maping_cfg[23:16] = 0x38;  
pri6 ->TC3, tcg_maping_cfg[31:24] =0x40。
```

4.6.4.4 调度权重配置举例

DWRR配置说明：pri0和pri1之间做DWRR调度，比例关系pri0：pri1=1:255，那么 pri0. pri_weight_cfg[7:0]=1， pri1. pri_weight_cfg[7:0]=255，最大包长为16,000 Byte的话，weight_offset_cfg[3:0]=14（其对应最大包长为16384Bytes）；

SP配置说明：pri2, pri 7之间做SP调度，调度器默认调度优先级为pri 0 > pri 1> pri 2 > pri 3> pri 4 > pri 5> pri 6 > pri 7，那么pri 2.tc_weight_cfg[7:0]=0， pri7.tc_weight_cfg[7:0]=0，最大包长为16,000 Byte的话，weight_offset_cfg[3:0]=14（其对应最大包长为16,384Byte），调度优先选择pri2，其次pri 7。

须知

- 调度器默认TC调度优先级为TC0 > TC1>TC2 > TC3> TC4 > TC5>TC6 > TC7。
- 调度器默认调度优先级为pri 0 > pri 1> pri 2 > pri 3> pri 4 > pri 5> pri 6 > pri 7。
- DWRR最小权重大于最大包长。

4.6.4.5 Shaping 配置

The shapers are implemented as a token bucket. A configurable number of tokens (called as Refill Size) are added into the token bucket at every configurable Refill



Period. When a packet is sent, the tokens of the packet size are removed from the token bucket.

Both CIR and PIR shapers can be calculated by the following formula.

$$xIR(Mbps) = \frac{xIR_B \times 2^{xIR_U} \times 8}{Tick \times 2^{xIR_S}} \times Clock\ Frequency(MHz)$$

Where,

- The text “xIR” represents either “CIR” or “PIR” .
- The CIR_B, CIR_U and CIR_S are configurable parameter of the CIR shaper.
- The PIR_B, PIR_U and PIR_S are configurable parameter of the PIR shaper.
- Tick is minimum refill period.
- The Clock Frequency is the clock frequency of ETS (1GHz) .
- Shaping accuracy is $\pm 1\%$.

Base on the above formula, the Refill Period of the CIR/PIR shaper can be calculated as follows:

$$Refill\ Period\ (clock\ cycles) = Tick \times 2^{xIR_S}$$

And the Refill Size of the CIR/PIR shaper can be calculated as follows:

$$Refill\ Size\ (bytes) = xIR_B \times 2^{xIR_U}$$

Both CBS and PBS of the shaper can be calculated by the following formula.

$$xBS\ (bytes) = xBS_B \times 2^{xBS_S}$$

Where,

- The text “xBS” represents either “CBS” or “PBS” .
- The CBS_B and CBS_S are configurable parameter of the CIR shaper.
- The PBS_B and PBS_S are configurable parameter of the PIR shaper.
- Warning:Suggest configure xIR_S the small the flow will be regular.

表 4-49 Shaping 配置

Level	Tick (cycles)
TC Shapers	48
TC Group Shapers	48
Port Shapers	6



须知

ETS调度中，由于目前通道缓存有限，当报文长度出现8K大包时，会出现调度比例不准的情况，甚至引发低优先级报文被调度的场景。

4.7 复位和初始化

4.7.1 强制复位

PPE逻辑异常，复位PPE，不复位MAC。

- 步骤1** 关闭TM的推包使能（TM.TM_PORT_EN.port_en 偏移地址0x2500），停止TM和软件向PPE发包。
- 步骤2** 关闭EGU的发送使能寄存器（MAG.IGU_EGU.MAC_EN_CFG.mac_tx_en，偏移地址0x050+0x400*mac_x4），并查询状态寄存器（MAG.IGU_EGU.MAC_EN_CFG.mac_tx_en_inner，偏移地址0x0050+0x400*mac_x4）为0。
- 步骤3** 配置PPE的MST停止向总线发送请求寄存器（MST_OOO的AM_CTRL_GLOBAL的bit0，偏移地址0x0000），将其配置为1。
- 步骤4** 查询PPE的MST安全复位指示寄存器（MST_OOO的AM_CURR_TRANS_RETURN的bit[1:0]，偏移地址0x0150），如果指示为全1，表示可以开始复位，否则隔20ms后再查询，如果连续10次查询为0，需要进一步判断逻辑是否异常或访问其他模块出现异常或长时间反压。

IGU和MAC要能处理sof和eof不匹配的报文。

须知

VF_NUM: 0~5; tnl_num: 0~25; port_num: 0~25; fw_grp_num: 0~15;
Pool_num 0~15

----结束



须知

这种复位会导致泄露Buffer，非必要不要使用。

POE为PPE的加包操作提供保序处理，POE根据添加包的数据结构中的Add_Id和Ostd_SN两个域段分别对PPE各通道进行保序处理。对于同一通道的加包操作（即Add_Id相同），POE按照Ostd_SN编号进行保序操作，并且通过寄存器POE_DISPATCH_CHN_WR_PTR可配置POE从哪个Ostd_SN编号开始处理，POE只有接收到对应的Ostd_SN编号的包任务后，才开始按照Ostd_SN编号递增的方式开始逐个处理，并依次按照Ostd_SN编号递增包顺序反馈总线响应。

所以复位PPE或POE时，需要同时配置保序记录信息，否则会导致PPE加包挂死：

1. 只要PPE的某个通道接收过报文后需要复位，都需要对POE的对应通道的保序记录信息进行清除。
2. 如果需要复位POE，则软件需要记录POE中当前的保序记录信息，复位POE后再将记录保序信息配置进去。
3. 建议软件使用粗暴的方法：如果复位PPE，则对POE的对应保序记录信息进行清0；如果复位POE，建议PPE一起复位。

POE记录的保序信息描述和清除方法如下：

寄存器POE_*_ORDER_PTR的默认值为0，即所有通道的默认从Tran_Index编号0开始处理。配置寄存器POE_DISPATCH_CHN_WR_PTR时，必须先将寄存器DEBUG_MODE_EN配置为0x5A5A，否则不能正确写寄存器POE_DISPATCH_CHN_WR_PTR。

4.7.2 安全复位/PF 复位

用于软件正常需求的复位，复位PPE，不复位MAC。

- 步骤1** 如果RoCE业务已经启动，请执行RoCE复位[步骤1](#)到[步骤5](#)。
- 步骤2** 关闭TM的推包使能（TM.TM_PORT_EN.port_en），停止TM和软件向PPE发包。
- 步骤3** 软件关闭BMU向PPE发送QOS反压信息使能寄存器（BMU.BMU_QOS_EN.bmu_ppe_bp_en 偏移地址0x12E0）。
- 步骤4** 关闭所有VF对应FW_TNL报文类型使能寄存器（MAM.MAM_CFG_FW_POOL.Pool_a_en、MAM.MAM_CFG_FW_POOL.Pool_b_en、MAM.MAM_CFG_FW_POOL.Pool_c_en、MAM.MAM_CFG_FW_POOL.Pool_d_en，偏移地址0x10200 + 0x4*fw_num）。
- 步骤5** 软件关闭MAM的申请Buffer使能（MAM.MAM_CFG_POOL_CFG.cfg_Pool_en，偏移地址0x10080 + Pool_num*0x4）。
- 步骤6** 确保DISPATCH模块连接PPE的端口相关通道请求已经清空，寄存器（DISPATCH.DISP_PORTn_STATE.*）。
- 步骤7** 先配置所有VF的复位请求寄存器为1（MAM.PPU_MAM_VF_FLUSH_REQ，偏移地址10B00），再配置安全复位指示（TOP.PPE_RST_REQ.rst_req 偏移地址0x0600）。
- 步骤8** TPU处理完当前报文（可以借助多模复位完成），并释放Buffer，输出完成指示，该步骤由硬件完成。
- 步骤9** RPU不处理新的SSU报文（可以借助多模复位完成），将RPU内残留的报文处理完，添加完POE，输出排空指示，该步骤由硬件完成。



- 步骤10** MAM看到RPU/TPU的排空指示后（TPU/RPU不会再向MAM发新的操作），排空总线操作后，输出PPE可以安全复位指示（TOP.PPE_RST_READY.rst_ready偏移地址0x0604），该步骤由硬件完成。
- 步骤11** 软件查询到逻辑的可以安全复位指示为1后，再按照VF读出MAM预申请的Buffer（MAM.MAM_BUFFER_ADDR_LOW 偏移地址0x10100 + Pool_num*0x8，MAM.MAM_BUFFER_ADDR_HIGH 偏移地址0x10104+ Pool_num*0x8），Buffer全部读出（读出Buffer地址为0表示该Pool的Buffer为空），查询（MAM.PPU_MAM_VF_FLUSH_DONE.ppu_mam_vf_flush_done，偏移0x10B04）为1表示所有Pool均读空。
- 步骤12** 关闭EGU的发送使能寄存器（MAG.IGU_EGU.MAC_EN_CFG.mac_tx_en，偏移地址0x050+0x400*mac_x4），并查询状态寄存器（MAG.IGU_EGU.MAC_EN_CFG.mac_tx_en_inner，偏移地址0x050+0x400*mac_x4）为0。
- 步骤13** 延时1微秒。
- 步骤14** 软件复位模块PPE，配置复位寄存器。
- ETH_Subctrl.SC_PPE_RESET_REQ0.* 偏移地址0xA10。
 - ETH_Subctrl.SC_PPE_RESET_REQ1.* 偏移地址0xA18。
 - ETH_Subctrl.SC_PPE_RESET_REQ2.* 偏移地址0xA20。
 - ETH_Subctrl.SC_PPE_RESET_REQ3.* 偏移地址0xA28。
 - ETH_Subctrl.SC_PPE_RESET_REQ4.* 偏移地址0xA48。
- 步骤15** 延时1微秒。
- 步骤16** 如果RoCE业务已经启动，请执行RoCE复位**步骤7**、**步骤8**，否则直接延时1微秒。
- 步骤17** 软件解复位PPE，配置解复位寄存器。
- ETH_Subctrl.SC_PPE_RESET_DREQ0.* 偏移地址0xA14。
 - ETH_Subctrl.SC_PPE_RESET_DREQ1.* 偏移地址0xA1C。
 - ETH_Subctrl.SC_PPE_RESET_DREQ2.* 偏移地址0xA24。
 - ETH_Subctrl.SC_PPE_RESET_DREQ3.* 偏移地址0xA2C。
 - ETH_Subctrl.SC_PPE_RESET_DREQ4.* 偏移地址0xA4C。

须知

软件查询：ETH_Subctrl.SC_PPE_RESET_ST0.* 偏移地址0x5A10、ETH_Subctrl.SC_PPE_RESET_ST1.* 偏移地址0x5A18、ETH_Subctrl.SC_PPE_RESET_ST2.* 偏移地址0x5A20、

ETH_Subctrl.SC_PPE_RESET_ST3.* 偏移地址0x5A28、ETH_Subctrl.SC_PPE_RESET_ST4.* 偏移地址0x5A48、ETH_Subctrl.SC_ETH_ROCEE_IRS_ST.* 偏移地址0x5A80各域段为0，表示时钟状态正常；超过10μs不为0，视为异常。

- 步骤18** 如果RoCE业务已经启动，请执行RoCE复位**步骤10**。

----结束



须知

POE为PPE的加包操作提供保序处理，POE根据添加包的数据结构中的Add_Id和Ostd_SN两个域段分别对PPE各通道进行保序处理。对于同一通道的加包操作（即Add_Id相同），POE按照Ostd_SN编号进行保序操作，并且通过寄存器POE_DISPATCH_CHN_WR_PTR可配置POE从哪个Ostd_SN编号开始处理，POE只有接收到对应的Ostd_SN编号的包任务后，才开始按照Ostd_SN编号递增的方式开始逐个处理，并依次按照Ostd_SN编号递增包顺序反馈总线响应。

所以复位PPE或POE时，需要同时配置保序记录信息，否则会导致PPE加包挂死；

只要PPE的某个通道接收过报文后需要复位，都需要对POE的对应通道的保序记录信息进行清除；

如果需要复位POE，则软件需要记录POE中当前的保序记录信息，复位POE后再将记录保序信息配置进去；

建议软件使用粗暴的方法：如果复位PPE，则对POE的对应保序记录信息进行清0；如果复位POE，建议PPE一起复位；

步骤6~步骤7以及步骤12~步骤18要求严格保序，要通过在每个操作间加isb()、dsb(sy)两条指令来确保操作不会乱序；

POE记录的保序信息描述和清除方法如下：

寄存器POE*_ORDER_PTR的默认值为0，即所有通道的默认从Tran_Index编号0开始处理。配置寄存器POE_DISPATCH_CHN_WR_PTR时，必须先将寄存器DEBUG_MODE_EN配置为5a5a，否则不能正确写寄存器POE_DISPATCH_CHN_WR_PTR。

4.7.3 VF 复位

某些场景下需要单模复位，步骤如下：

- 步骤1** 完成TM该VF的复位，保证不再有该VF的任务输入PPE，否则会造成Buffer泄露；保证CPU加包接口不再有该VF的任务加载，否则会造成Buffer泄露；如果RoCE业务已经启动，请执行RoCE复位**步骤1**到**步骤5**。
- 步骤2** 关闭VF对应FW_TNL报文类型使能寄存器（MAM.MAM_CFG_FW_POOL.Pool_a_en、MAM.MAM_CFG_FW_POOL.Pool_b_en、MAM.MAM_CFG_FW_POOL.Pool_c_en、MAM.MAM_CFG_FW_POOL.Pool_d_en，偏移地址0x10200 + 0x4*fw_num）。
- 步骤3** 关闭对应VF向BMU申请使能寄存器（MAM.MAM_CFG_POOL_CFG.cfg_Pool_en，偏移地址0x10080 + Pool_num*0x4）。
- 步骤4** 配置需要复位通道的模式复位寄存器（MAM.PPU_MAM_VF_FLUSH_REQ.ppu_mam_vf_flush_req，偏移0x10B00）为1。
- 步骤5** 轮询PPP所有分发表项（L2 TCAM、自定义0~3和L3 TCAM），找到该VF对应的所有表项（由FW_TNL反映射推知VF），配置PPP对应的VF的FW_TNL分发表项的动作为丢弃，阻止报文继续往RPU送，详细请参考表项配置流程请参见“[各个表项配置流程](#)”。
- 步骤6** 查询复位通道的模式复位完成寄存器（MAM.PPU_MAM_VF_FLUSH_DONE.mam_vf_flush_done，偏移0x10B04），对应VF位为1后表示，该VF的任务报文已清除完成。
- 步骤7** 回收缓存，读取需要复位VF对应的16个Pool预申请的缓存寄存器（MAM.MAM_BUFFER_ADDR_LOW 偏移地址0x10100 + Pool_num*0x8，



MAM.MAM_BUFFER_ADDR_HIGH 偏移地址 $0x10104 + \text{Pool_num} \times 0x8$), Buffer全部读出(读出Buffer地址为0表示该Pool的Buffer为空), 再查询(MAM.PPU_MAM_VF_FLUSH_DONE.ppu_mam_vf_flush_done, 偏移 $0x10B04$)为1表示所有Pool均读空。

步骤8 关闭模式工作寄存器, 配置工作寄存器(TOP.PPE_VF_FUNC_ENABLE.vf_func_enable 偏移地址: $0x10200$)为0。

步骤9 配置需要复位通道的模式复位寄存器(MAM.PPU_MAM_VF_FLUSH_REQ.ppu_mam_vf_flush_req, 偏移 $0x10B00$)为0。

步骤10 如需配置MAM的VF级寄存器, 参考MAM初始化步骤, 最后打开对应VF向BMU申请使能寄存器(MAM.MAM_CFG_POOL_CFG.cfg_Pool_en, 偏移地址 $0x10080 + \text{Pool_num} \times 0x4$)。

步骤11 打开VF对应FW_TNL报文类型使能寄存器(MAM.MAM_CFG_FW_POOL.Pool_a_en、MAM.MAM_CFG_FW_POOL.Pool_b_en、MAM.MAM_CFG_FW_POOL.Pool_c_en、MAM.MAM_CFG_FW_POOL.Pool_d_en, 偏移地址 $0x10200 + 0x4 \times \text{fw_num}$)。

步骤12 在分发表中(L2 TCAM、自定义0~3和L3 TCAM)恢复PPP对应的VF的FW_TNL表项的动作为正常, 详细请参考表项配置流程“[各个表项配置流程](#)”。

步骤13 打开模式工作寄存器, 配置工作寄存器(TOP.PPE_VF_FUNC_ENABLE.vf_func_enable 偏移地址: $0x10200$)为1。

须知

所有的单模复位都不会复位相应制式的寄存器的配置值、状态与统计值, 如果该VF重新被分配, 残留的某些寄存器有影响, 需要软件在重新分配前将这些寄存器清除或配置正确。

----结束

4.7.4 初始化

4.7.4.1 PPE 复位后初始化流程

步骤1 软件复位模块PPE, 配置复位寄存器。

- ETH_Subctrl.SC_PPE_RESET_REQ0.* 偏移地址 $0xA10$ 。
- ETH_Subctrl.SC_PPE_RESET_REQ1.* 偏移地址 $0xA18$ 。
- ETH_Subctrl.SC_PPE_RESET_REQ2.* 偏移地址 $0xA20$ 。
- ETH_Subctrl.SC_PPE_RESET_REQ3.* 偏移地址 $0xA28$ 。
- ETH_Subctrl.SC_PPE_RESET_REQ4.* 偏移地址 $0xA48$ 。

步骤2 延时1微秒。

步骤3 打开PPE时钟, 配置寄存器。

- ETH_Subctrl.SC_PPE_ICG_EN0.* 偏移地址 $0x310$ 。
- ETH_Subctrl.SC_PPE_ICG_EN1.* 偏移地址 $0x318$ 。



- ETH_Subctrl. SC_PPE_ICG_EN2.* 偏移地址0x320。
- ETH_Subctrl. SC_PPE_ICG_EN3.* 偏移地址0x328。
- ETH_Subctrl. SC_PPE_ICG_EN4.* 偏移地址0x330。

如果使能ROCEE模块，打开顶层打拍逻辑时钟。

(ETH_Subctrl. ICG_EN_ETH_ROCEE_IRS.* 偏移地址0x368)。

说明

软件查询寄存器：

- ETH_Subctrl. SC_PPE_ICG_ST0.* 偏移地址0x5310。
- ETH_Subctrl. SC_PPE_ICG_ST1.* 偏移地址0x5318。
- ETH_Subctrl. SC_PPE_ICG_ST2.* 偏移地址0x5320。
- ETH_Subctrl. SC_PPE_ICG_ST3.* 偏移地址0x5328。
- ETH_Subctrl. SC_PPE_ICG_ST4.* 偏移地址0x5330。
- ETH_Subctrl. ICG_EN_ETH_ROCEE_IRS.* 偏移地址0x5368。

各域段为1，表示时钟状态正常；超过10μs不为1，视为异常。

步骤4 延时1微秒。

步骤5 软件解复位PPE，配置解复位寄存器。

- ETH_Subctrl. SC_PPE_RESET_DREQ0.* 偏移地址0xA14。
- ETH_Subctrl. SC_PPE_RESET_DREQ1.* 偏移地址0xA1C。
- ETH_Subctrl. SC_PPE_RESET_DREQ2.* 偏移地址0xA24。
- ETH_Subctrl. SC_PPE_RESET_DREQ3.* 偏移地址0xA2C。
- ETH_Subctrl. SC_PPE_RESET_DREQ4.* 偏移地址0xA4C。

如果使能ROCEE模块，配置顶层打拍逻辑解复位。

ETH_Subctrl. SC_ETH_ROCEE_IRS_RESET_DREQ.* 偏移地址0xA84。

说明

软件查询寄存器：

- ETH_Subctrl. SC_PPE_RESET_ST0.* 偏移地址0x5A10。
- ETH_Subctrl. SC_PPE_RESET_ST1.* 偏移地址0x5A18。
- ETH_Subctrl. SC_PPE_RESET_ST2.* 偏移地址0x5A20。
- ETH_Subctrl. SC_PPE_RESET_ST3.* 偏移地址0x5A28。
- ETH_Subctrl. SC_PPE_RESET_ST4.* 偏移地址0x5A48。
- ETH_Subctrl. SC_ETH_ROCEE_IRS_ST.* 偏移地址0x5A80。

各域段为0，表示时钟状态正常；超过10μs不为0，视为异常。

关闭PPE的MST OOO的读写使能（寄存器TOP. AM_CFG_PORT_RD_EN和TOP. AM_CFG_PORT_WR_EN配置为0，偏移地址0x2018和0x201C）。

步骤6 等待查询PPE的MST OOO的读写状态为0（ TOP. AM_CURR_PORT_STS，偏移地址0x2100 ）。

步骤7 配置Memory初始化启动寄存器。

- TPU. PPE_TPU_MEM_START_INIT. mem_start_init 偏移地址 0x0300。



- RPU. PPE_RPU_MEM_START_INIT. rpu_mem_start_init 偏移地址 0x00C0。
- MAM. MAM_MEM_START_INIT. mem_start_init 偏移地址 0x0720。
- SSU. SSU_MEM_INIT_START. mem_init_start 偏移地址 0x0208。
- PPP. PPP_MEM_INIT_CTRL. ppp_mem_init_start 偏移地址 0x0428)。

步骤8 查询Memory初始化完成寄存器。

- TPU. PPE_TPU_MEM_INIT_DONE. mem_init_done 偏移地址 0x0304。
- RPU. PPE_RPU_MEM_INIT_DONE. rpu_mem_init_done偏移地址 0x00C4。
- MAM. MAM_MEM_INIT_DONE. mem_init_done偏移地址 0x0724。
- SSU. SSU_MEM_INIT_DONE. mem_init_done偏移地址 0x020C。
- PPP. PPP_MEM_INIT_STATUS. ppp_mem_init_done 偏移地址 0x0430) 为1时，表示初始化完成，否则查询等待。

步骤9 打开PPE的MST OOO的读写使能（寄存器TOP.AM_CFG_PORT_RD_EN和TOP.AM_CFG_PORT_W_EN配置为0xFFFF，偏移地址0x 2018和0x 201C）。

步骤10 配置公共寄存器：

- TOP. SR_IOV_CTL.* 偏移地址0x0108。
- TOP. SR_IOV_VF_OFFSET.* 偏移地址0x012C。
- TOP. PPE_PORT_RATE_MODE.* 偏移地址0x1100+ port_num*0x4。
- TOP. PPE_CFG_FW_GRP_MAP.* 偏移地址0x1140+fw_grp_num*0x4。

其余寄存器如不需要可不配置。

步骤11 配置各模块寄存器，包括MAG.IGU_EGU.*，PA.*，PPP.*，SSU.*，MAM.*，RPU.*，TPU.*，其中部分寄存器可以选择性配置。

步骤12 配置各VF业务寄存器（TOP. PPE_VF_FUNC_ENABLE.* 偏移地址0x10200），模块进入正常工作状态。

----结束

须知

1. **步骤1~步骤7**要求严格保序，要通过在步骤间加isb()、dsb(sy)两条指令来确保操作不会乱序。
2. PA中IGU_CRC_CK_EN寄存器配置需要和MAC中的FCS使能相关联，配置需保持一致。
3. 上电复位完成后，加载Bios系统时，需要将ETH_Subctrl. SC_SYSCTRL_PPE_PROT（偏移地址：0x2A58）配置成0x3，确保只有安全操作可以访问Xsec寄存器。默认值可供测试场景时使用。

4.7.4.2 PA 寄存器配置流程

步骤1 基于端口配置端口类型配置寄存器0x1048~0x1060。

步骤2 基于端口配置解析使能配置寄存器0x1000 。



- 步骤3** 根据业务需求配置L2/L3/L4对应的自定义解析模板配置寄存器0x2C~0x18C，寄存器默认值和标准协议匹配。
- 步骤4** 如果需要解析隧道报文，需要根据业务需求配置隧道解析模板配置寄存器0x8~0x24。
- 步骤5** 基于端口VLAN解析配置寄存器0x1004~0x1008 和VLAN 选择配置寄存器0x103C 。
- 步骤6** 配置TC映射配置寄存器100c~102c。
- 步骤7** 如果使能L2/L3/L4自定义报文提取功能，需要基于端口配置自定义报文提取配置寄存器0x1030~0x1038。

----结束

4.7.4.3 TPU 寄存器配置流程

步骤1 配置公共寄存器。

- TPU. PPE_TPU_WR_BD_OUTSTAND_CTRL.* 偏移地址 0x0420。
- TPU. PPE_TPU_ENDIAN_CTRL.* 偏移地址0x0430+tnl_num*0x4。
- TPU. PPE_TPU_OUTSTAND_CTRL.* 偏移地址 0x0500+tnl_num*0x4。
- TPU. PPE_TPU_PORT_TNL_SIZE.* 偏移地址0x0580+tnl_num*0x4。
- TPU. PPE_TPU_VF_CORE_ADD_MAP_TAB.* 偏移地址 0x0600+vf_num*0x4。

步骤2 配置访问总线寄存器。

- TPU. PPE_SMMU_USER_STRMID.* 偏移地址 0x0F00+vf_num*0x4。
- TPU. PPE_DDR_SMMU_USER.* 偏移地址 0x0F20+vf_num*0x4。
- TPU. PPE_SMMU_USER_SSET.* 偏移地址 0x0F60。
- TPU. PPE_CFG_BUS_QOS.* 偏移地址 0x0F64。
- TPU. PPE_CFG_BUS_CACHE.* 偏移地址 0x0F68。

其余的模板寄存器需要按照相应业务选择性配置。

----结束

4.7.4.4 MAM 初始化流程

步骤1 配置PF级寄存器：

从MAM_CFG_POOL_MAP（偏移地址0x0100）到MAM_VF_VIS_BMU_OFFSET（偏移地址0x02A8）所有的寄存器。

步骤2 配置业务级的寄存器，由VF来依次配置：

1. 配置自适应和大小端寄存器，MAM_CFG_BUF_ADPT_EN（偏移地址0x10020）。
2. 配置FW_TNL对应的Pool号和使能，MAM_CFG_FW_POOL（偏移地址0x10200+fw_num*0x4）。
3. 配置FW_TNL对应的其他属性，MAM_CFG_FW_ACC（0x10600+fw_num*0x4）。
4. 配置Pool对应的属性和向BMU申请Buffer开关，MAM_CFG_POOL_CFG（0x100zh-cn_topic_000000187781514580+Pool_num*0x4）。



说明

如果分步骤配置此寄存器，需要最后配置向BMU申请Buffer开关。

----结束

须知

- 1.每个Pool是FIFO或LIFO模式不可动态配置，切换配置后需要复位PPE。
- 2.当BMU Buffer带有维测头（N个Cacheline）时，PPE使用的Buffer地址为原始地址偏移N个Cacheline，PPE中buf_size为BMU实际分配空间大小扣除N个Cacheline。
3. MAM_CFG_BUFF_FIFO_THRSLD不可以大于MAM_CFG_FIFO_CFG.cfg_fifo_depth，否则会出现Buffer泄露。

4.7.4.5 RPU 初始化流程

步骤1 配置PF级寄存器：

- 配置正常报文向POE推送POE描述符的地址相关寄存器
RPU_POE_CHNL0_ADDR_L/H（偏移地址0x0240~0x0244）、
RPU_POE_CHNL0_STRIDE（偏移地址0x0248）、RPU_POE_CHNL0_VF_OFFSET
（偏移地址0x024C）、配置推送POE描述符的source_port值寄存器
RPU_POE_SOURCE_PORT_CFG（偏移地址0x0514+tnl_num*0x40）。
- 配置1588回传时间戳向POE推送POE描述符的地址寄存器
RPU_POE_TS_ADDR_L/H（偏移地址0x0330~0x0334）、配置推送POE描述符的
source_port/vqi值寄存器RPU_TS_PD_DATA（偏移地址0x0350+mac_id*0x04）。
- 配置调试网口相关的寄存器：使用哪些ppe_tnl作为调试网口
RPU_TEST_PORT_CFG（偏移地址0x0BC0）、调试网口超时中断时间门限
RPU_TEST_PORT_TIMEOUT（偏移地址0x0520+tnl_num*0x40）、调试网口上报
正常中断需要接收的报文个数RPU_TEST_PKT_CNT_CFG（偏移地址
0x0524+tnl_num*0x40）。

步骤2 配置业务级的寄存器，由VF来依次配置：

配置FW_TNL对应的其他属性，MAM_CFG_FW_ACC（该配置寄存器在MAM中，如果已配置，请忽略该步骤，偏移地址0x10600+fw_num*0x4）。

----结束

4.7.4.6 PPP 初始化流程

配置PPP相关寄存器和表项，请按照PPP表项间接访问配置。

步骤1 配置直通表，对应的MAC端口和PPE_TNL口走LSW还是直通模式。

步骤2 如果需要用到流表，首先配置tcam分配：

- PPP_FD_LSW_TCAM_ALLC_MODE_0、PPP_FD_LSW_TCAM_ALLC_MODE_1、
PPP_FD_LSW_TCAM_CFG、
- PPP_FD_FW_TCAM_ALLC_MODE_0、PPP_FD_FW_TCAM_ALLC_MODE_1和
PPP_FD_FW_TCAM_MODE。



- 配置查表使能，PPP_PORT_MISC_CFG对应的*_lkup_en。
- 配置流表的模板匹配表，PPP_FD_KEY_TEMPL_DATA_0~3和PPP_FD_KEY_TEMPL_CFG。
- 配置流表的模板key选择，PPP_FD_KEY_SEL_TDATA_*、PPP_FD_KEY_SEL_MDATA和PPP_FD_KEY_SEL_CFG。
- 配置流表的key和AD，PPP_FD_KEY_DATA_*和PPP_FD_KEY_CFG。

步骤3 配置L2表，按照间接读写方式添加key。

- 如果场景需要，配置L2表的mask，PPP_L2_MAC_LSB_MASK、PPP_L2_MAC_MSB_MASK。
- 需要学习和漂移检测，配置PPP_L2_TX_SA_CFG和PPP_L2_RX_SA_CFG。

步骤4 配置VLAN表和MGID表，可做VLAN入口检查和出口过滤，多播复制等。

步骤5 配置LSW入口属性PPP_ING_IV_VLAN_CFG_0、PPP_ING_IV_VLAN_CFG_1、PPP_ING_DEF_VLAN_TAG、PPP_ING_ATTR_CFG等。

步骤6 配置LSW转发和过滤属性，PPP_FILTER_FORWD_CFG、PPP_*_ISO_CFG_*、PPP_FILTER_FORWD_CFG_OP，端口隔离需要配置，否则会过滤掉报文。

步骤7 配置LSW出口属性，是否剥离VLAN，PPP_EGR_PORT_ATTR和PPP_EGR_PORT_ATTR_CFG。

步骤8 如果场景需要，配置trunk相关PPP_TRUNK_*。

步骤9 配置RSS相关寄存器，PPP_RSS_CRC_CFG_VAL、PPP_RSS_CRC_MATRIX_CFG和PPP_RSS_KEY_MASK_0~12。

步骤10 配置自定义分发表，PPP_SELF_DEF_DISP*。

步骤11 配置优先级和QoS相关。

----结束

4.7.4.7 SSU 初始化流程

步骤1 （可选）根据具体使用场景配置通道入口缓存空间。配置以下寄存器：

- 接收方向：SSU.SSU_RX_SHARE_BUFFER.*，SSU.SSU_RX_SHARE_THR.*。
- 发送方向：SSU.SSU_TX_SHARE_BUFFER.*，SSU.SSU_TX_SHARE_THR.*。

步骤2 （可选）根据高/低优先级单/多播Buffer需求，配置以下寄存器：

SSU.SSU_LO_PRI_SGL_BUF_SIZE.*、SSU.SSU_LO_PRI_MULT_BUF_SIZE.*、
SSU.SSU_HI_PRI_MULT_BUF_SIZE.* 以及丢弃水线
SSU.SSU_LO_PRI_SGL_DROP_THR.*、SSU.SSU_LO_PRI_MULT_DROP_THR.*、
SSU.SSU_HI_PRI_MULT_DROP_THR.*。

步骤3 （可选）根据用户场景需求配置SSU内部的BUFFER及水线及流控寄存器。

- SSU.SSU_TX_DUP_OT_PORT_DROP_THR.*。
- SSU.SSU_RX_DUP_OT_PORT_DROP_THR.*。
- SSU.SSU_TX_OT_PORT_DROP_THR.*。
- SSU.SSU_RX_OT_PORT_DROP_THR.*。

- SSU. SSU_TX_OQ_DROP_THR.*。
- SSU. SSU_RX_OQ_DROP_THR.*。
- SSU. SSU_TX_DUP_OT_PORT_DROP_CELL_THR.*。
- SSU. SSU_RX_DUP_OT_PORT_DROP_CELL_THR.*。
- SSU. SSU_TX_OT_PORT_DROP_CELL_THR.*。
- SSU. SSU_RX_OT_PORT_DROP_CELL_THR.*。
- SSU. SSU_TX_OQ_DROP_CELL_THR.*。
- SSU. SSU_RX_OQ_DROP_CELL_THR.*。
- SSU. SSU_RX_REV_PD_THR.*。
- SSU. SSU_TX_REV_PD_THR.*。

说明

以上寄存器非必要可不配置。

步骤4 配置VLAN编辑相关寄存器。

- SSU. SSU_RX_VLAN_TYPE_CFG.*。
- SSU. SSU_TX_VLAN_TYPE_CFG.*。

步骤5 针对不同速率场景配置ETS相关寄存器。

- SSU.ETS_TC_WEIGHT.*、SSU.ETS_TCG_WEIGHT.*、SSU.ETS_WEIGHT_OFFSET.*。
- SSU.ETS_TCG0_MAPING.*、SSU.ETS_TCG1_MAPING.*。
- SSU.ETS_TCG2_MAPING.*、SSU.ETS_TCG3_MAPING.*。
- SSU. ETS_PORT_SHAPING.*、SSU.ETS_TCG_MAPING.*。

步骤6 针对不同速率场景配置多播调度相关寄存器。

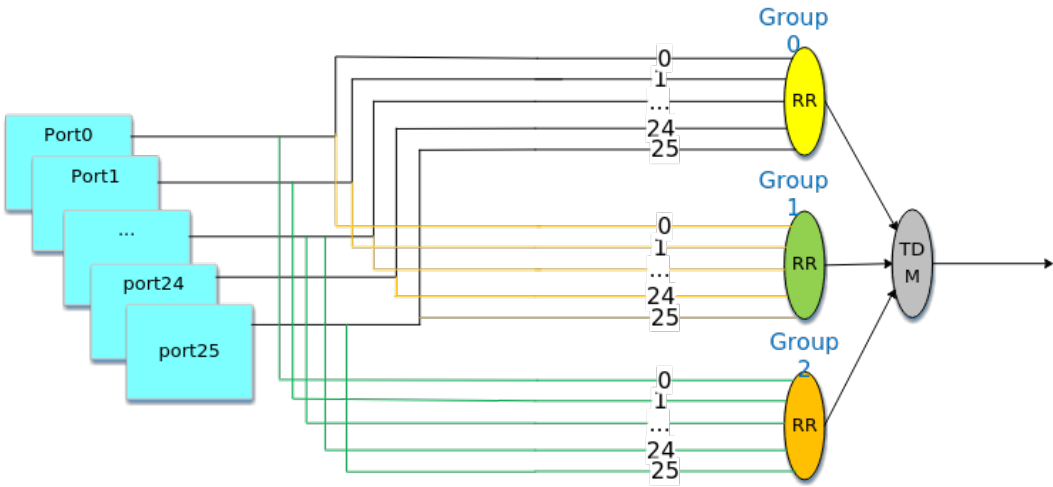
- 如果使用DUP WRR，首先配置SSU. SSU_DUP_WRR_WEIGHT.wrr_weight_offset为6，再根据业务要求配置SSU. SSU_DUP_WRR_WEIGHT.*寄存器中三个优先级报文的比例关系。
- 如果使用多播SHAPING，请根据业务需求配置以下寄存器：
 - SSU. SSU_HI_PRI_MUL_SHAPING.*。
 - SSU. SSU_LO_PRI_MUL_SHAPING.*。
 - SSU. SSU_LO_PRI_UNI_SHAPING.*。

说明

当WRR和SHAPING都有配置时，在复制小包总性能（pps）达到瓶颈出现阻塞时，调度效果以WRR比例优先，当对应优先级总性能（bps）达到SHAPING配置值时以SHAPING调度效果为主。

步骤7 根据各端口速率配置SSU发送侧调度归属group相关寄存器。配置原则如图4-67所示。

图 4-67 根据各端口速率配置 SSU 发送侧调度归属 group 相关寄存器原则



网口的分布原则：

1. GE固定分配在Group2。
注：如果没有GE时，Group2也可以分配给其他速率使用。
2. 其他的网口分布的场景如表4-50所示：黄色为group0，绿色为group1，其他为group2。

表 4-50 网口分布场景

场景	XGE、XAUI		GE/2.5G
0	2	2	x
1	0	0	x

端口归属group以bitmap形式呈现，根据上表4-50分别配置SSU_EG_GROUPx_PORT_BITMAP选择端口归属。

须知

1. SSU完成初始化所有步骤完成前不允许TPU/PA有报文进入SSU否则会导致SSU内部异常。
2. 步骤1配置时，如果有RoCE业务场景，Tx方向配置的Buffer大小要适当调整，TX \RX所有Buffer的总大小尽量不要超过1.5MByte。避免出现不符合预期的丢包。
3. 配置端口号超出范围这种异常SSU无保护，要软件做好保护才行，不然SSU内部会出现异常。

----结束

4.7.5 LSW alone

PPE支持CPU子系统独立复位时，LSW和仍能正常工作；在CPU子系统解除复位之后，PPE可以重新将报文发往CPU子系统；此操作需要整芯片配合操作，在PPE的操作步骤如下：

- 步骤1** 系统准备CPU子系统复位，置起sys_abnormal_flag告知PPE准备CPU子系统复位。
- 步骤2** PPE收到告知后，会反压TM，等待500ms，保证所有TPU接收到的报文都已经发送完毕。
- 步骤3** 配置CRG（参考CRG配置流程）进行CPU子系统复位，并且复位PPE内部子模块MAM，TPU，RPU，TX_PA。
- 步骤4** 进入CPU子系统复位状态，此时PPE的LSW交换通路，网口到网口的交换数据流不受影响。
- 步骤5** 复位完毕准备回复CPU子系统数据流，配置CRG（参考CRG配置流程）撤销PPE内部子模块MAM，TPU，RPU，TX_PA对应复位。
- 步骤6** 初始化PPE内部子模块MAM，TPU，RPU，TX_PA，并完成相关业务配置。
- 步骤7** 配置SSU credit初始化使能寄存器SSU_RPU_PORT_INIT_ENABLE、SSU_ROCEE_PORT_INIT_ENABLE为0x03FF_FFFF 恢复SSU与Rocee、RPU的credit机制。
- 步骤8** 配置系统控制寄存器撤销sys_abnormal_flag；此时数据流开始发往CPU子系统。

----结束

4.8 芯片调试 DFX 功能特性描述

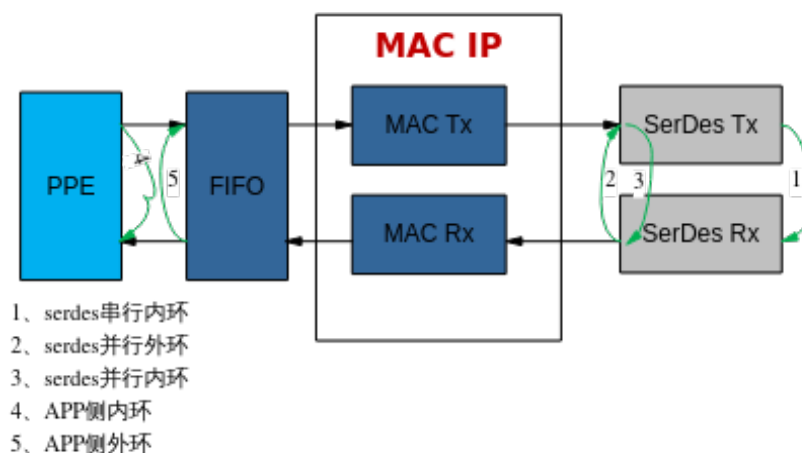
4.8.1 可靠性

暂无。

4.8.2 环回

MAG只支持如图4-68环回4，5。

图 4-68 MAG 环回





说明

环回只支持调试端口收发是否通流，不支持线速，不支持其他功能。

4.8.3 模块维测

模块级维测信息请参见寄存器（专用地址段，此处不述）

须知

对于MAC中的MIB统计访问，软件需要保证两次访问（包括读-写，读-读，写-读，写-写）至少256个XXVGE时钟（Hi309a 按照400MHz），否则会出现访问异常。

4.8.3.1 统计

原则：各模块丢弃报文均可统计；

各模块统计守恒（即入口报文=丢弃报文+出口报文）；

各模块可以互相配合完成统计。

1. TM推送发送描述符地址FIFO中的发送首描述符地址个数。
2. CPU推送发送描述符地址FIFO中的发送首描述符地址个数。
3. 接收到MAC的报文统计。
4. 成功写入接收描述符的报文个数统计（包括截断报文、及接收向BMU进行丢弃的报文）。
5. 成功接收且添加POE成功的报文统计。
6. 因接收FIFO满整包丢弃的统计。
7. 因接收FIFO满包截断的统计。
8. 接收多模禁止后丢弃的报文统计。
9. 被写入发送描述符地址个数（包括发送描述符地址FIFO溢出没有成功写入的统计）。
10. 发送多模禁止后丢弃的报文统计。
11. 成功发送的报文统计。
12. 发送FIFO空而导致报文截断的统计。总线延时过大导致、长度域填为0、mem失效都会出现该统计，可以通过中断排除是哪种错误导致的。
13. 发送计算checksum失败的报文统计（发送报文过长）。
发送计算checksum最大包长为9.6K
14. 接收、发送状态机当前状态。
15. 接收、发送FIFO状态。
16. 发送报文基于通道、VF的统计。
17. 接收报文基于通道、VF的统计。



4.8.3.2 告警上报

1. 发送描述符SRAM 1bit ECC错误的地址（记录第一次的）。
2. 发送描述符SRAM 2bit ECC错误的地址（记录第一次的）。
3. 存储缓存的SRAM ECC 2bit错误地址（记录第一次的）。
4. 存储缓存的SRAM ECC1bit错误地址（记录第一次的）。
5. 存储Group配置值的SRAM出现2bit ECC错误地址（记录第一次的）。
6. 存储Group配置值的SRAM出现1bit ECC错误地址（记录第一次的）。
7. 存储qid配置值的SRAM出现2bit ECC错误地址（记录第一次的）。
8. 存储qid配置值的SRAM出现1bit ECC错误地址（记录第一次的）。
9. 存储报文类型解析的SRAM出现2bit ECC错误地址（记录第一次的）。
10. 存储报文类型解析的SRAM出现1bit ECC错误地址（记录第一次的）。
11. 存储Group和Pool的映射表SRAM出现2bit ECC错误地址（记录第一次的）。
12. 存储Group和Pool的映射表SRAM出现1bit ECC错误地址（记录第一次的）。
13. CPU是否推送发送描述符成功，bit[31:0]对应推包接口31~0是否成功。
14. 畸形报文检查。
15. 内部FIFO溢出。
16. 存储Group的mem连续出现1bit ECC错误，且来不及回写。
17. 存储qid的mem连续出现1bit ECC错误，且来不及回写。
18. 存储报文解析方式的mem2连续出现1bit ECC错误，且来不及回写。
19. 存储报文解析方式的mem1连续出现1bit ECC错误，且来不及回写。
20. 存储Pool信息的mem连续出现1bit ECC错误，且来不及回写。

4.8.3.3 中断上报

1. 发送报文描述符的SRAM出现2bit ECC错误中断。
2. 发送报文描述符的SRAM出现1bit ECC错误中断。
3. 丢弃多Buffer报文中断。
4. 工作描述符状态机心跳超时检测中断。
5. 解析状态机心跳检测超时中断。
6. 接收状态机心跳检测超时中断。
7. 发送状态机心跳检测超时中断。
8. 发送描述符地址FIFO溢出中断。
9. 读写DDR总线错误中断。
10. 接收报文因优先级低丢弃中断（要使用的Pool未使能时也会上报该中断）。
11. 接收报文因没有缓存丢弃中断。
12. 接收报文在接口模块中丢弃中断，丢弃的原因有2种可能：FIFO满、报文超过规定的长度。
13. 发送报文因FIFO空丢弃中断。
14. 发送描述符报文长度域为0中断。
15. 发送报文的SRAM出现奇偶校验错误中断。
16. 接收报文的SRAM出现奇偶校验错误中断。



17. 接收的报文满足配置条件（达到包个数上限或者没达到但接收时限到，一般用于调试网口）的正常中断。
18. 释放缓存失败中断，只有链式buf释放缓存到bmu失败（bmu的响应为非ok，达到最大尝试次数）时才会报，针对单链表释放不会有失败的情况（返回非ok也认为释放成功了）。
19. 存储Y.1731统计的SRAM出现奇偶校验错误中断。
20. 存储Group和Pool的映射表SRAM出现2bit ECC错误中断。
21. 存储Group和Pool的映射表SRAM出现1bit ECC错误中断。
22. 存储报文类型解析的SRAM出现2bit ECC错误中断。
23. 存储报文类型解析的SRAM出现1bit ECC错误中断。
24. 存储qid配置值的SRAM出现2bit ECC错误中断。
25. 存储qid配置值的SRAM出现1bit ECC错误中断。
26. 存储Group配置值的SRAM出现2bit ECC错误中断。
27. 存储Group配置值的SRAM出现1bit ECC错误中断。
28. 存储缓存的SRAM出现2bit ECC错误中断。
29. 存储缓存的SRAM出现1bit ECC错误中断。
30. vf Pool（或资源Pool）和物理Pool映射失败原始（基于vf）。
31. 需要使用的Pool未使能丢包原始中断寄存器（基于vf）。
32. 缓存指针溢出失败原始中断寄存器（基于vf），保存接收缓存的fifo溢出。
33. 向BMU申请buf失败原始中断寄存器（基于vf），一直回no_buf，并达到最大尝试次数。

4.8.3.4 消息上报

4.8.3.5 Debug

本模块提供了以下方式用于内部状态查询，以方便系统调试（Debug）。

- 本模块提供给了Debug Bus接口，该接口可配置和可读取RAM寄存器配置信息及统计信息。
- L3 Cache内部的所有寄存器，除了L3_STATUS以外，均可以通过Debug Bus接口进行读写访问。

4.8.3.6 注意事项

暂无。

4.9 故障处理指南

4.9.1 软硬件接口故障

- 命令格式硬件不支持。
- 命令参数超界。
- 响应超时未响应。



4.9.2 硬件内部数据完整性错误

- 控制信息的Parity、ECC 1bit/2bit错误。
- 数据信息的Parity、ECC 1bit/2bit错误。
- 静态配置表项的Parity、ECC1bit/2bit错误。

4.10 典型问题定位流程

无。

4.11 时钟复位

- 工作时钟在400MHz时钟频率，复位策略异步复位。
- PPE上电时复位默认处在“软复位”状态。
- PPE上电时钟默认是“使能”状态。
- 此外，子模块PA、SSU、PPP、RPU、TPU、MAM上电时复位默认处在“软复位”状态。

4.12 PPE_TNL 的性能指标

4.12.1 单 TNL 单 BD 性能

单个网口一个报文仅有一个BD的场景：PPE支持以3.8Mpps/10Gbps进行报文收发，即长度330Bytes以上的报文性能达到10Gbps，长度330Bytes以下的报文性能达到3.8Mpps。

4.12.2 单 TNL 多 BD 性能

多BD报文性能：是对应长度的单BD报文性能的至少80%。

- 对接收侧而言，此时要求可用的Buffer长度（除去skip）都是1.5kB以上。
- 对发送侧而言，此时要求最大3个节点，除尾片外，其他片典型包长1536B。

4.12.3 多 TNL 单 BD 性能

所有通道同时工作时，所有通道支持的同时收发流量是20Gbps，不超过总流量时，任意通道满足带宽。

须知

需要考虑DDR自刷新，每7800ns需要自刷新一次，自刷新时间消耗300ns+，自刷新期间会阻塞操作，导致延时变大。

总线会乱序返回操作结果，OOO保序后返回，EDA需要覆盖。

309a考虑到都是使用BMU的Buffer，不会动态管理页表，SMMU的配置都是用2M或更大Buffer，所以性能测试时不考虑访问操作出现页表MISS。



4.12.3.1 转发性能

1. MAC侧之间转发长度为64Byte及以上的报文支持44Gbps线速收发。
2. CPU侧之间转发长度为330Byte及以上的报文，且一个报文仅有一个BD的支持20Gbps线速收发。

4.12.4 吞吐率（Throughput）

无。

4.12.5 延迟（Latency）

处理延迟需求：

包长：500Byte，309a(Tx)->309a(Rx)serdes口环回端到端时延：

Latency MIN（空载）：1.5 μ s

Latency MAX（DMA引擎为空，但内部FIFO数据Buffer满）：3 μ s



5 SA 加速器

5.1 概述

SA硬件加速IP（SA）是进行模式匹配的IP模块。SA 硬件加速IP获取到报文后，从内存中读取相应的模式匹配指令，对报文进行匹配，匹配完成后上报匹配结果。

说明

本文档分为加速器和规则转换器两部分，其中规则转换器为软件。

5.2 关键术语

表5-1为与SA相关的关键术语表。

表 5-1 SA 加速器术语表

术语	描述
SA	Service Awareness，业务感知。
AMBA	Advanced Microcontroller Bus Architecture，由ARM公司研发的一种SoC片上系统总线，可将RISC处理器集成在其他IP芯核和外设中。
BD	Buffer Descriptor，描述符，包括发送描述符和接收描述符，描述报文在主存中的存储信息，包括报文起始地址，报文解析结果等等。
Graph	图，对应规则子库。
PPM	packet process manager 报文处理模块，实现报文和报文的预处理。
CPM	cross process manager 跨包处理模块，实现跨包信息的读取和存储，以支持流模式匹配。
BMF	bloom filter 分组过滤模块。



术语	描述
PMU	pattern match unit 模式匹配单元。
RPM	result process manager 结果处理和上报模块。

5.3 特性

SA 硬件加速IP有如下特性：

- 支持纯字符串和正则模式匹配。
 - 支持10万条模式串。
 - 支持以单条模式串为粒度指定大小写敏感，不支持在单条模式串内分段改变大小写敏感属性。
 - 支持一个报文同时在最多不超过4个graph（lib子库）内进行匹配查找。
 - 最大支持126个子库，子库id从1-126（0和127不允许使用）。
 - 支持32bit位宽的模式串id。
- 支持流模式和块模式匹配。
 - 支持块模式匹配，单个报文结束则匹配结束，只上报命中的首尾位置均位于本报文内的结果。
 - 支持65535Byte以内的长报文处理，但匹配扫描范围不超过4095Byte。
 - 支持同一条匹配流的报文之间进行跨包匹配，且是否跨包可由软件可配，但只允许在新建流时改变。
 - 跨包长度软件可配（64-2048Byte多档位），默认256Byte，但只允许在新建流时改变该配置。
 - 同一条流的匹配任务保序返回（假设依赖—CPU将归属到同一条流的匹配任务发到相同的POE队列），不同流（或者块模式）匹配任务不保序（即便来自同一POE队列）。
 - 支持最大32M条并发流跨包匹配。
- 模式匹配结果预处理。
 - 单个报文最多可上报1023个命中结果，在此范围内可由软件在下发匹配任务时修改此数量门限，超过配置门限以外的那部分命中结果将被丢弃。
 - 命中结果包含lib_id（即子库id）+rule_id（即模式串id）+start_offset+end_offset，其中start_offset为命中起始位置到本报文第一个字节的相对偏移地址，支持命中起始位置位于上一报文的情况。
 - 单个匹配任务内同一条模式串多次命中的场景下，提供4种去重后上报的方式：
 - 命中子库id+模式串id相同即认为结果相同，只上报其中一次命中信息。
 - 命中子库id+模式串id和命中起始位置都相同即认为结果相同，只上报其中一次命中信息。
 - 命中子库id+模式串id和命中结束位置都相同即认为结果相同，只上报其中一次命中结果。
 - 命中子库id+模式串id、命中起始位置和命中结束位置都相同才认为结果相同，只上报其中一次命中信息。



- 以上方式由软件针对单个模式串指定。
 - 支持对同一个报文内的多个命中结果按命中位置排序（从小到大）后上报，可通过任务描述符指定按start_offset（命中起始位置）或按end_offset（命中结束位置）排序。
- 性能特性。
 - 100K HTTP页面报文测试，硬件性能达到7.5Gbps。
 - 21K HTTP页面模型下（打开应用识别）硬件匹配性能达到3.75Gbps。
 - 21K HTTP页面报文测试，在负荷不超过80%性能规格的条件下，异步匹配和同步匹配的平均处理延时分别小于60μs和20μs。
- 知识库加载和更新。
 - 知识库加载时，知识库片内表项加载完毕后允许在加载某个分库指令表的同时进行其他分库的匹配任务处理。
 - 不支持多线程加载，驱动必须对加载流程加锁。
 - 提供知识库表项的回读接口。
- 接口特性。
 - 符合AMBA4.0协议。
 - 提供一个AXI Master 接口，数据位宽256bits，地址位宽48bits。
 - 提供一个AXI Slave 接口，用于对匹配引擎进行配置。
 - SA加速器通过GET_POE获取模式匹配任务，通过ADD_PACKET反馈任务完成。
 - 提供高低两种优先级调度机制，以适配产品的同步匹配和异步匹配两种应用场景，但要求软件对同一条匹配流的优先级指示必须保持一致。
- 业务级DFX。
 - 延时测量功能，对从POE获取匹配任务到向POE反馈匹配任务完成的端到端延时进行测量，支持按指定lib、同步、异步三个维度分别测量平均延时、最小延时、最大延时，软件可通过配置寄存器来启动或关闭该功能。
 - 支持提前中止单个报文的匹配的，匹配中止的条件按匹配时间和伪命中次数两个维度分别进行设置，同步和异步匹配独立配置。
 - 对入口报文流量按高低优先级分别提供帧级和字节级计数。
 - 对不能正常匹配处理（匹配超时中止、匹配信息溢出、结果去重异常、结果数量超规格、伪命中次数超门限等）的报文在上报结果中体现错误类型，并进行统计，以便产品进行场景分析和配置优化。
 - 对同步匹配报文增加处理延时监控，超过延时门限的打上响应错误标记后返回。
- 中断上报。
 - 所有RAM支持ECC校验，出错则上报中断，需要软件回写的RAM记录错误地址；配置表项支持1bit纠错回写（可配）。
 - 所有fifo提供上下溢中断。
 - AXI访问返回异常中断上报，记录错误类型。
 - 对读入描述信息和知识库表项等进行合法性检查。
 - 报文不能正常匹配处理（匹配超时中止、结果去重异常、匹配中间信息缓存溢出等）时上报中断。
 - 加速器内部数据通道异常（帧级sop和eop不匹配、内部描述符和报文不一致等）时上报中断。

- 所有中断都有对应的原始中断寄存器、中断使能、中断注入寄存器、相关的中断错误信息寄存器。
- 异常中断通过三根中断线（三个中断级别）上报，信号类型为脉冲。
- 芯片DFX。
 - 分组过滤模块对bmf_hash表命中、bmf链表返回、bmf链表命中中进行统计。
 - 各精确匹配引擎对需匹配报文、匹配扫描group_id数、匹配命中数等进行统计。
 - 结果上报模块对去重数、去重异常等进行统计。
 - 关键状态机状态可见，并做不跳转超时监控。
 - 内部缓存（fifo/ram）状态可见。
 - 提供读写系统内存的pps级统计和MByte级统计。

5.4 遵循协议

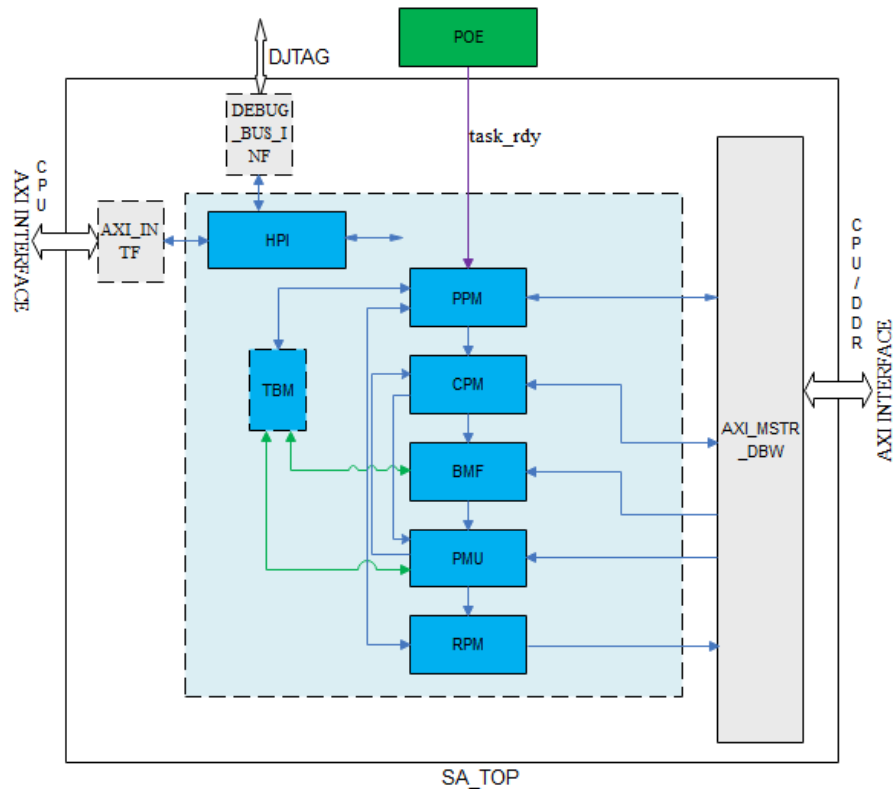
总线接口遵循AMBA4.0协议。

5.5 功能描述

5.5.1 SA 加速器应用图

SA IP在整个HI309a芯片中的应用如图5-1所示。

图 5-1 SA 加速器连接示意图



 说明

: POE模块, HI309a内部的包保序模块, 和SA IP进行相关数据交互。



: SA IP模块区域。

SA硬件加速IP主要完成模式匹配功能。连接示意图中各数据通路介绍如下:

- POE通过带外信号通知SA有task需要读取。
- SA从POE获取task数据。
- SA从PPE存储buf中获取SA描述符。
- SA从内存中读取软件自定义的buf_info信息。
- SA从内存中读取报文。
- SA从内存中读取流表, 获取跨包相关信息。
- SA从内存中读取跨包缓存(同一条流的上一报文)。
- SA从内存中读取bmf_link表项, 获取分组ID和key。
- SA从内存中指令表, 进行精确匹配。
- SA匹配完成后, 将匹配结果写入内存指定区域。
- SA通知POE任务完成, 释放报文缓存。
- SA匹配完成后刷新跨包缓存。
- SA匹配完成后刷新跨包流表。

5.5.2 字符串匹配与正则匹配

匹配模式串里只包含纯字符, 无其他正则符号, 举例如下:

- 模式串: vmall\x2ecom。
- 报文: www.vmall.com。
- 报文命中模式串, 符号"."的ASCII值为0x2E。

匹配模式串里包含正则符号, 不能按纯字符串对待, 举例如下:

- 模式串: \wmall\x2ecom。
- 报文1: www.tmall.com。
- 报文2: www.vmall.com。
- 报文1和2都能命中模式串, \w表示英文字母、数字或下划线。

5.5.3 块模式匹配和流模式匹配

块模式匹配以单个报文(或单个匹配任务)为边界, 超出本报文范围后不再匹配。流模式会将同一条匹配流的报文连成一体来匹配检查。举例如下:

- 模式串1: http://{0,32}\x2esohu\x2ecom
- 同流的两个顺序报文之一: http://www.s
- 同流的两个顺序报文之二: ohu.com/music



5.5.4 模式匹配结果上报处理

- 支持最大1023个rule_id+start_offset+end_offset上报，在此范围内可由软件在下发匹配任务时修改此数量门限，当超过配置门限时RPM丢弃多余的规则ID并上报丢弃标记。
- 对上报的匹配结果进行去重操作。
- 匹配结果写入系统内存后通过POE给CPU返回任务完成消息。

5.5.5 知识库加载功能

参见初始化流程中的“[知识库加载](#)”章节。

5.5.6 SA 获取 task (GET_POE)

软件给SA分配了task读地址，配置到SA的寄存器中：
REG_PPM_GET_POE_ADDR_L32B_CFG和REG_PPM_GET_POE_ADDR_H16B_CFG，每次去读取task时，SA向该地址发送AXI读请求，POE根据该读操作地址译码得到当前GET_POE操作的加速器编号，从而得到当前加速器的GET_POE操作能获取的物理Group组，通过内部调度，得到读取的任务反馈给SA加速器。

说明

POE通过写寄存器方式告知SA有task可以获取，寄存器有值时，SA发起task的读操作，POE在队列为空时将寄存器清零，SA检测到寄存器值为0，不再发起task读操作。因此寄存器值从有效到无效有一定滞后性，SA可能会多发起task读操作，对于多读的task，POE将task返回数据置无效。举例：如果POE中的task个数为4个，SA发起了5个task读操作，则POE返回给SA的task中，前4个是有效的task数据，第5个task数据是无效的，通过task数据结构中的Node_Code来指示，当Node_Code为7'h7F时表示task数据无效。

5.5.7 SA 添加 task (ADD_PACKET)

SA通过ADD_PACKET命令添加包任务到POE中。

SA添加包任务：

- 软件给SA分配了task写地址，配置到SA的寄存器中：
REG_PPM_ADD_POE_ADDR_L32B_CFG和REG_PPM_ADD_POE_ADDR_H16B_CFG，每次反馈task完成时，SA向该地址发送AXI写请求，POE根据该写操作地址译码得到当前ADD_POE操作的加速器编号，将任务添加到对应的物理队列中。
- Hi309a项目通过credit机制保证加速器添加的包任务不会超过POE的接收能力，POE缓存加包任务，并自行处理真满假满问题，不会再给加速器反馈真满假满。
- SA添加包任务前，先判断credit用量计数是否小于最大可用值，若不满足则等待，满足后才能发起加包操作。每发起一次加包操作，credit用量计数加1，POE返回credit后减去返回的credit数量。

5.5.8 连续字符丢弃 (8+8) 功能

SA硬件引擎支持在某些测试场景下，对于大量连续出现的相同字符，对其匹配的部分结果按一定规则进行丢弃，以提高此场景下的处理能力。

此功能按报文粒度可配，若需打开，将BD中的same_cut字段置为1即可；若不需要请置为0。



丢弃规则为，如果总线上报文的第n拍（1拍为32Byte，即内部总线位宽，按32Byte粒度丢弃）数据，和第n-1拍，及第n+1拍上的相邻8Byte数据，都是相同的字符，则将第n拍输出的结果丢弃，即使有匹配结果也不输出。

如图5-2所示，0x80这拍数据，和0x60这拍数据的相邻8Byte（Byte24~31），及0xA0这拍数据的相邻8Byte（Byte0~7），都是相同字符“*”，所以在这拍数据扫出的结果会被丢弃掉。同理可知，0xA0、0xC0扫出的结果也会被丢弃掉。

图 5-2 连续字符丢弃示意图

32字节（总线宽度）																																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x000	h	t	t	p	s	:	/	/	i	d	l	.	c	l	o	u	d	.	h	u	a	w	e	i	.	c	o	m	/	C	A	S
0x020	/	p	o	r	t	a	l	/	l	o	g	i	n	A	u	t	h	.	h	t	m	l	?	v	a	l	i	d	a	t	e	d
0x040	=	t	r	u	e	&	t	h	e	m	e	N	a	m	e	=	r	e	d	&	s	e	r	v	i	c	e	=	h	t	t	p
0x060	s	%	3	A	%	2	F	%	2	F	w	w	.	v	m	a	l	l	.	*	*	*	*	*	*	*	*	*	*	*	*	*
0x080	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x0A0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x0C0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x0E0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x100	2	5	2	F	%	2	5	2	F	w	w	.	v	m	a	l	l	.	c	o	m	%	2	5	2	F	&	l	o	g	i	
0x120	n	C	h	a	n	n	e	l	=	2	6	0	0	0	0	0	0	0	&	r	e	q	C	l	i	e	n	t	T	y	p	e
0x140	2	6	&	l	a	n	g	=	z	h	-	c	n																			

5.5.9 结果去重功能

SA硬件引擎在输出结果前，对于同一个匹配任务内同一条模式串多次命中的情况，提供结果去重处理。

如图5-3所示，去重原理描述如下：

每个结果使用其对应的结果信息{soff_crs, lib_id, rule_id, start_offset, end_offset}（不关注部分置成全1）计算哈希值，将哈希值作为地址去查找哈希表（总空间为4096），若未能在哈希表中找到该结果，且哈希表中该地址的8个空间未用满，说明该结果之前未出现过，上报，并在哈希表对应地址空间中顺序存入该结果，供后续比较使用；如果能在哈希表中找到该结果，则说明该结果之前已上报过，为重复结果，直接丢弃，实现去重。

若未能在哈希表中找到该结果，但哈希表中该地址的8个空间已用满，说明其他结果的哈希值和该结果相同（即哈希冲突），并占满了哈希表的空间。为了应付哈希冲突，设置了空间为8的逃生桶，在哈希表满时可使用逃生桶空间继续存储，其用法和哈希表类似。注意，逃生桶为所有哈希地址共用。

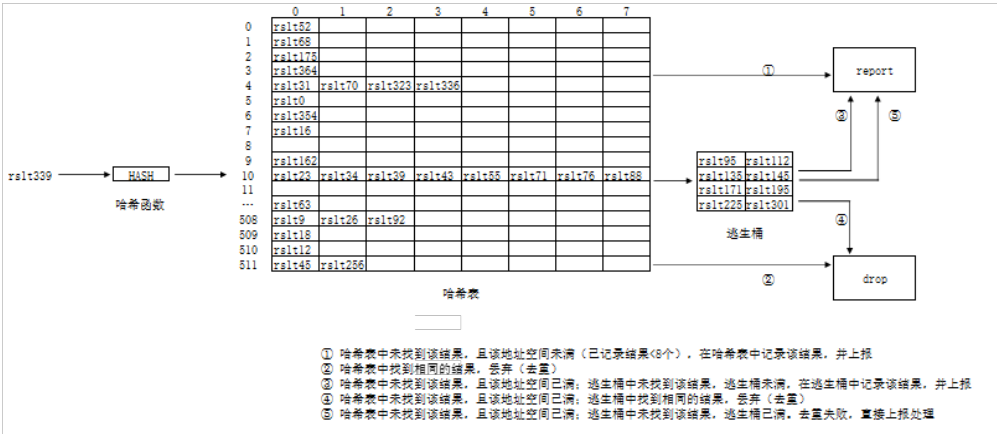
若哈希表和逃生桶中都未能找到该结果，且哈希表和逃生桶空间都已用满，此时该结果直接上报，但无法保证不会和之前上报的结果重复，会打上去重失败标记供软件特殊处理，并产生去重失败告警。

上述去重失败场景在结果数量较多（哈希表填充约40%及以上）时较容易出现；在测试场景刻意构造较多（超过16个必然出现）哈希值相同但内容不同的结果时也能触发此情况出现。

每个匹配任务的结果处理完后，哈希表和逃生桶都会被清空，不同任务之间的去重处理不会相互影响。



图 5-3 去重原理示意图



5.6 编程接口

5.6.1 SA 加速器数据结构

交互数据结构参见如表5-2、表5-3、表5-4、表5-5。

表 5-2 SA BD（cpu 发给加速器的任务描述符）数据结构 64bytes

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0								
bd_type				new_flowid																									
buf_num/ tb_data_type					nge_rst _cnt				c r s _ e n	crs_le n_sel			rsv					next_vqi											
buf_info_addr/pkt_addr/tb_src_addr[31:0]																													
pkt_len													buf_info_addr/pkt_addr/ tb_src_addr[47:32]																
match_rslt_addr/tb_des_addr[31:0]																													



lib_id3	lib_id2	lib_id1	lib_id0
crs_addr[31:0]/tb_crc32			
rsv		crs_addr[47:32]	
ctrl_blk_addr[31:0]			
ctrl_blk_addr[63:32]			
bd_crc16		pkt_crc16	
rsv			
rsv			
rsv			
rsv			
说明			
bd_type	0：匹配任务BD； 1：rsv，无效。 2：表项写BD，表示本描述符用于知识库表项加载，对应有效字段为：new_flag、tb_src_addr[47:0]、tb_len、nge_rst_cnt、tb_data_type、tb_lib_id、tb_des_addr[31:0]、last_flag。 3：表项读BD，表示本描述符用于知识库表项回读，对应有效字段为：new_flag、tb_src_addr[47:0]、tb_len、nge_rst_cnt、tb_data_type、tb_des_addr[31:0]。		
new_flag	匹配任务BD： 新流标记，表示一条新流的首个报文 表项读写BD： 每段表项读写的第一个BD置1，表项按tb_data_type和tb_lib_id（从知识库输出结构体中提取）不同来分段。		
flowid	流号，硬件匹配引擎用于区分报文属于哪一条流的，据此完成跨包信息的获取。规格为32M条流，因此只取低25bit有效		
buf_num	buf_info的个数		
tb_data_type	知识库的表项类型，对应到规则转换器输出结果中的data_type，data_type值范围0~63。注意，此字段和buf_num共用，知识库加载和回读时仅支持单buf写入或读取，不支持buf_info向量的形式。		



crs_en	<p>跨包使能标志，对该流所有报文有效；软件必须保证在一条匹配流存续期间，本字段保持不变。</p> <p>此字段与blk_mode字段有区别，blk_mode只是用于指示永远不使用跨包匹配的业务处理，此字段用于需要跨包匹配的业务动态打开或关闭跨包功能。</p>
crs_len_sel	<p>跨包长度选择，对该流所有报文有效；0表示64Byte，1表示128Byte，2表示256Byte，3表示512Byte，4表示1024Byte，5表示2048Byte；其他采用默认值256Byte。软件必须保证在一条流存续期间，本字段保持不变。</p>
next_vqi	<p>任务反馈使用的Vqi号（代替之前的next_group、next_qid），SA完成任务后发送到POE的Add_Packet里的Vqi域需要跟本域相同。</p>
pkt_addr/ buf_info_addr/ tb_src_addr	<p>匹配任务BD：</p> <p>复用域，若link_flag==0，则pkt_addr有效，本域表示报文存放的首地址；否则，buf_info_addr有效，本域表示buf_info的存储首地址。</p> <p>表项写BD：</p> <p>tb_src_addr有效，表示待加载表项数据在内存中存放的首地址；硬件引擎将从该地址读取数据后写入引擎内部存储。</p> <p>表项读BD：</p> <p>tb_src_addr有效，表示待读取表项数据在硬件引擎内部的偏移地址；地址取值方式为 $0x0 + n * pkt_len$，为了设计简单化，要求软件对每段待加载表项都以2KByte为粒度（小于2KByte则以实际为准）进行加载，因此该地址在一段表项范围内都是从0开始以2KByte为粒度递增。</p>
pkt_len	<p>匹配任务BD：</p> <p>复用域，若link_flag==0，则本域表示报文长度；否则，本域表示多个buf_info描述的报文总长度。</p> <p>表项读写：待读写或搬移的数据长度。</p>
match_rslt_addr/ tb_des_addr	<p>匹配任务BD：</p> <p>match_rslt_addr有效，匹配结果存放区域的首地址，匹配结果的数据格式见下方表格。</p> <p>表项写BD：</p> <p>tb_des_addr有效，表示待加载表项数据将要写入到硬件引擎内部的偏移地址；地址取值方式为 $0x0 + n * pkt_len$，为了设计简单化，要求软件对每段待加载表项都以2KByte为粒度（小于2KByte则以实际为准）进行加载，因此该地址在一段表项范围内都是从0开始以2KByte为粒度递增。</p> <p>表项读BD：</p> <p>tb_des_addr有效，表示硬件引擎将从内部读出的表项数据写入内存区域的首地址，硬件引擎从内部读出表项数据后将其写入本地地址开始的内存区域。</p>
link_flag	<p>用于指示本报文是否有buf_info描述具体报文存储信息，本字段仅当bd_type为匹配任务BD时有效。</p>



same_cut	1有效，用于控制在相同字符连续出现且hash命中时，是否对中间部分结果进行丢弃。按之前产品反馈，模式串用于协议识别时不能设为有效。（逻辑内部设计上需将该字段传给BMF）
sort_type	用于指示匹配引擎上报结果时的排序方式。为1表示按end_offset排序；为0表示按start_offset（包含sof_crs）排序。
blk_mode	块模式匹配标志，用于指示该次匹配为块模式匹配，对此匹配任务硬件匹配引擎无需读取/刷新流表和跨包缓存。 增加此标志是为了扩展对无需跨包匹配业务的支持，该种匹配业务处理不分配流号，不建立流表。
last_flag	匹配任务BD： 本字段无效。 表项读写BD： 每段表项读写的最后一个BD置1，表项按tb_data_type和tb_lib_id（从知识库结构体中获取）不同来分段。
max_rslt_num	最大结果个数，限制匹配返回的结果个数不能超过此值；软件根据当前内存空间大小动态分配，但不能配置为0。
lib_id0~3	本报文需要查找匹配的子库ID号（0x1到0x7E），共4个字段。必须从lib_id0到lib_id3顺序使用，不使用的字段务必置为0。举例：假设某个报文需要同时在子库x和子库z中同时查找，则软件必须将lib_id0置为子库x，lib_id1置为子库z，lib_id2和lib_id3置为0。 注意：这里使用的lib_id跟NGE传给SA规则转换器时一样，必须为物理lib_id（不能是虚拟lib_id）。
tb_crc32	仅当bd_type为表项写BD且last_flag为1时本字段才有效；驱动在某段表项加载的最后一个BD时将该段表项的CRC值写入本字段，硬件引擎对其进行校验，若不吻合则认为表项加载出错。
crs_addr	跨包缓存首地址，由软件分配，供硬件引擎读写访问。
ctrl_blk_addr	控制块地址，软件控制块地址暂存区，SA不使用，软件收到匹配完成消息后使用该地址获取控制消息，该字段只在匹配任务BD中有效。
pkt_crc16	仅当bd_type为匹配任务时本字段才有效。驱动将发送给硬件引擎的报文计算CRC16（多项式按照硬件手册约定）后填入此字段，硬件引擎读取报文后校验。本字段仅用于软硬件开始联调时的问题定位，实际发货的软件版本为了性能考虑而不会计算此CRC值，因此硬件引擎必须提供配置寄存器关掉校验。
bd_crc16	对所有类型的BD有效。驱动将本BD的所有字段（不包括bd_crc16）计算CRC16（多项式按照硬件手册约定）后填入此字段，硬件引擎读取BD后校验。本字段仅用于软硬件开始联调时的问题定位，实际发货的软件版本为了性能考虑而不会计算此CRC值，因此硬件引擎必须提供配置寄存器关掉校验。
rsv	保留区，值为0

表 5-3 buf_info 结构

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
pkt_addr[31:0]																														
pkt_len															pkt_addr[47:32]															
说明																														
pkt_addr		buf_info对应报文的地址																												
pkt_len		buf_info对应报文的长度																												

表 5-4 匹配结果格式

[illegible]

rsv			
rule_id_0			
start_offset_0		end_offset_0	
rsv	sof - crs - 0	rsv	lib_id_0
.....			
.....			
.....			
rule_id_n			
start_offset_n		end_offset_n	
rsv	sof - crs - n	rsv	lib_id_n
pad (若Word16~Word_18+3n的长度非64B整数倍, 则填充至64B整数倍)			
-			
说明			
rslt_num	结果个数		
rslt_type	rlt_type: 3'b000: 匹配失败; 3'b001: 匹配成功;		
cut_flag	截断标志, 表示已命中的规则数超过了允许上报的个数, 多余的匹配结果信息会被丢弃; 如本错误发生频繁, 建议将匹配任务描述符中的结果个数门限调大, 但需注意相应的内存占用也会增大。		



err_info	<p>错误信息，每bit表示一种错误类型，高有效。</p> <p>[0]：rsv。</p> <p>[1]：rsv。</p> <p>[2]：rsv。</p> <p>[3]：结果去重异常，上报结果里可能存在重复命中信息。</p> <p>[4]：匹配超时中止，命中结果数量可能比实际少。</p> <p>[5]：匹配中间信息溢出，命中结果数量可能比实际少。</p> <p>[6]：规则扫描范围超出跨包长度，命中结果数量可能比实际少。</p> <p>[7]：hash假命中次数超门限，命中结果数量可能比实际少，如本错误发生频繁，建议将hash假命中数门限调大，但可能引起性能下降。</p> <p>[8]：流相关错误——流表基地址未配置就收到该流范围内的匹配任务；流表无效时收到new_flag无效的匹配任务。</p> <p>[9]：libid异常，包括出现大于126和全为0两种情况。</p> <p>其它bit暂时保留。</p>
magic_word	魔术字，固定为十六进制数abcd，用来判断SA是否往该内存写过结果
rule_id	命中的规则id，该id在子库范围内编号。
lib_id	命中的子库编号。
sof_crs	sof跨包标志，表示sof不在当前的报文内，落在上一个报文内。
end_offset	规则结束的偏移，表示与规则最后一个字符对齐的报文字符的下标，报文字符的下标从0开始使用。
start_offset	当sof_crs为0是，表示当前报文与规则首字符匹配的报文字符的下标，当sof_crs为1时，表示从当前报文的0字节开始往上一个报文反向的字符下标。
pad	为避免非64B cacheline对齐写入，系统先读64B拼接后再写造成性能损失，SA要求跨包缓存、匹配结果缓存基地址按64B cacheline对齐，并且大小也要为64B整数倍，SA写入时都按64B cacheline对齐写入，以提高性能。

表 5-5 ADD_PACKET 数据结构 0x41（带 Affinity，Timestamp，Prefetch 或 Hardware 扩展场景）

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									



Lo ca tio n		Vq i_R sv	Vqi[9:0]			SourcePort[7:0]		T_Len[3:0] =4'b0100	T_TagV alue[3:0] =4'b0001	M sg _L en =2 'b01
R s v	O r d e r _B i n d V a l	Order_B indID	Or de r_ Ty pe	Order_TagValue[23:0]						
Addr[31:0]										
Re l	R l s _I n d	Re s e r v e d	Addr _Rsv	Pref_Of fset1	Pref_Of fset2	Addr[47:32]				
Reserved										
Reserved										
Hac_Occ_Timestamp_Value								T_Len[3:0] =4'b0001	T_TagV alue[3:0] =4'b0010 or 4'b0011 or 4'b0111 or 4'b1000 (Timest amp)	



Hac_Occ_CRC	Hac_Occ_Add_ID	Hac_Occ_Ost_d_SN	Hac_Occ_DFX_VFID	Hac_Occ_FRD_Qos	T_Len[3:0] =4'b0001	T_TagValue[3:0] =4'b0100 (Hardware)
说明						
POE基础字段说明		CPU GET POE接口返回的消息长度为配置队列的位宽（256bit和512bit），当队列位宽大于Msg_Len位宽时，GET返回的Msg_Len长度之外的域段为全0；HAC GET POE接口返回的消息长度固定为256bit；（注：在HAC接口里面AffInd,Pref_Offset1,Pref_Offset2域段无效）PUSH_MSGQ场景 POE根据Msg_len判断推送256bit还是512bit给MSGQ；				
Msg_Len		指示变长消息长度，单位128bit，（0:128bit，1:256bit，2:384bit，3:512bit）--SA收到的固定为2'b01，否则丢弃处理				
T_TagValue[3:0](Word0)		数据结构按照TLV方式定义，类型标识T_TagValue=4'b0001，POE任务 --SA收到的固定为4'b0001，否则丢弃处理				
T_Len[3:0](Word0)		数据结构按照TLV方式定义，单位32bit，结构长度T_Len=4'b0100，128bits任务--SA收到的固定为4'b0100，否则丢弃处理				
SourcePort[7:0]	SourcePort[6:0]: Node_Code	7位任务源设备号： 7'b00000000 ~ 0x7'b11111110（0x0 ~ 0x7E）：任务源设备号； 7'b11111111（0x7F）：无效源设备号，写无效；当读操作返回该值，表示获取任务失败（即POE内无有效的任务）。--SA通过此字段判断任务是否有效				
	SourcePort[7]:	保留用作规格扩展				
Vqi[9:0]		Vqi替代原先的（Group, Qid），POE使用VQI作为VF的VQI路由表的相对偏移，访问路由表获取虚拟Qid和虚拟GROUP，并完成虚拟Group到物理Group的转换--SA约定Vqi[2:0]为Qid，即根据Vqi[2:0]查询得到队列属性（该属性通过寄存器REG_PPM_QID_PRI_CFG配置，0-低优先级业务队列，1-高优先级业务队列，2-表项下发队列），从而决定这些队列的报文在SA内部的处理。注意：Vqi和POE内部队列的对应关系也是可配的，软件要保证配到POE的队列属性和配到SA的保持一致。				
Vqi_Rsv		保留用作规格扩展				



Location	目的POE的编码，用来识别消息目的地是否跨Die/片，若本DIE POE非目的POE，则将任务转发给目的POE，对应POE寄存器POE_FORWARD_POE_DIE_ID；如果Location与POE_FORWARD_POE_DIE_ID配置值相同，即该消息是给本Die POE，否则转发给跨Die POE。bit[1]: Reserved bit[0]: 0 本片；1 跨片；（各模块要求做成可配置，配置方式同Vqi）注意：POE实现是拿Location域段和POE寄存器配置的值进行匹配，匹配上则认为为本地的；在Hi1383V100仅支持1个POE，约束要求POE寄存器必须配置为“00”。
Order_TagValue[23:0]	24位流号。
Order_Type	保序类型：
	■ 2' b00: NULL;
	■ 2' b01: ORDER;
	■ 2' b10: ATOMIC serial;
	■ 2' b11: ATOMIC parallel.
Order_BindID	保序任务绑定的LCPU的绑定单元ID，在推送LCPU私有队列应用中，POE为每个LCPU提供10个绑定单元，支持每个LCPU同时绑定10个保序任务，LCPU在处理完毕保序任务后，需要在保序命令中携带Order_BindID；（注意：该域段ADD POE时软件不需要填写，已Reserved掉；GET POE时POE会输出该域段）（注：在HAC接口里面该域段无效）
Order_BindVal	保序任务绑定指示：（注意：该域段ADD POE时软件不需要填写，已Reserved掉；GET POE时POE会输出该域段） （注：在HAC接口里面该域段无效）
	■ 1' b0: 非保序任务；
	■ 1' b1: 绑定了保序任务，LCPU在处理完毕保序任务后，需要在保序命令中携带Order_BindID；
Addr[47:0]	报文接收描述符地址（共48位）
Pref_Offset1/2	Stash预取字段，支持预取“Addr”偏移offset的Cacheline。“0”表示不预取，“1”表示预取Offset为0的Cacheline（64Byte），“2”表示预取Offset为1的Cachline（64Byte），以此类推。Pref_Offset1/2各对应一条Cacheline的偏移。
Addr_Rsv	保留用作规格扩展
Reserved(Word3)	保留用作规格扩展
Rls_Ind	0x0: 不需要释放Buffer，不回写描述符；0x1: 需要释放Buffer，POE/TM调用BMU的释放接口透传Rel字段



Rel	释放标志位，指示释放方式。 注1：只有当PD的扩展TLV指示为PPE发送描述符时，回写标志才有效；否则为不回写； 注2：PPE释放和回写时，不看rls_ind指示；POE和TM释放和回写时，优先看Rls_Ind;因为PPE的实现约束，要求“由软件保证rls_ind为0时，Rel=2'b11且发送任务描述符中回写Wb=0”。	回写		释放		
		w b= 0	w b= 1	发送描述符地址（首片且无报文）	发送描述符地址（含首片 / 非首片 / 有报文 / 无报文）	报文地址
	2'b00：只需要释放addr对应的单Buffer，不需要读描述符，不需要回写状态；	N	N	Y	N	N
	2'b01：需要释放描述符和报文的Buffer，根据描述符指示位决定是否回写状态（当PPE发送方向指示为回写时，不释放发送描述符首片节点的buffer，其余节点的buffer都需要正常释放）； 注意：1：如果wb=1，且为非分离式buffer（发送描述符首片和报文在一个buffer），则不能释放；2：如果wb=0，且为非分离式buffer（发送描述符首片和报文在一个buffer），释放时只能释放1次，要避免描述符和报文在同一buffer被重复释放两次；	N	Y	N	Y ^{注1,2}	Y
	2'b10：只需要释放报文的Buffer，不释放描述符buffer，根据描述符指示位决定是否回写状态；注意：如果wb=1，且为非分离式buffer（发送描述符首片和报文在一个buffer），则该buffer不能释放；	N	Y	N	N	Y
	2'b11：不释放，根据描述符指示位决定是否回写状态。	N	Y	N	N	N



Reserved(Word4/5)	保留用作规格扩展。
T_TagValue[3:0](Word6)	数据结构按照TLV方式定义，类型标识T_TagValue=4'b0001，POE任务 --若Word6的{T_Len[3:0]，T_TagValue[3:0]}为0x12或0x13或0x17或0x18，即表示Word6为时间戳信息，处理方式为T_Len[3:0]和T_TagValue[3:0]透传，Hac_Occ_Timestamp_Value填下述信息；否则Word6为Reserved，ADD POE时填0 T_TagValue 4'b0010：Telemetry专用Timestamp，所有加速器透传（除TM，TM不透传，TM计算出口时戳和描述符中时戳的差值；）4'b0011：Timestamp入口时间戳 4'b0111：Timestamp出口时间戳 4'b1000：Timestamp入口和出口的时戳之差
T_Len[3:0](Word6)	数据结构按照TLV方式定义，单位32bit，结构长度T_Len=4'b0100，128bits任务-见T_TagValue[3:0](Word6)的描述。
Hac_Occ_Timestamp_Value	POE支持统计报文/消息在POE内的latency，时戳精度为0.1us，计数时长1.678s，宽度为2124Bit。--见T_TagValue[3:0](Word6)的描述。
T_TagValue[3:0](Word7)	不需处理
T_Len[3:0](Word7)	不需处理
Hac_Occ_FRD_Qos	不需处理，透传
Hac_Occ_DFX_VFID	不需处理，透传
Hac_Occ_Ostd_SN	不需处理
Hac_Occ_Add_ID	不需处理
Hac_Occ_CRC	不需处理

CRC计算方法如下：

```
//-----
//1、配表数据计算CRC方法
//1) 配表数据进行32B对齐，padding数据填0x0。
//2) seed设置为0xffffffff，调用sa_crc32(unsigned int seed,const unsigned char *buf,unsigned int len)进行计算；有多个2KB时，再次调用sa_crc32()时，seed值为上次CRC的32bit返回值按位取反。
//-----
/*
* Table for the AUTODIN/HDLC/802.x CRC.
*
```



```
* Polynomial is
*
*  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 +$ 
*  $x^5 + x^4 + x^2 + x + 1$ 
*/
const uint32_t crc32table_le[256] =
{
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419,
0x706af48f, 0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4,
0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07,
0x90bf1d91, 0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856,
0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4,
0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
0x325b5a8fa, 0x42b2986c, 0xdbb9c9d6, 0xacbcf940, 0x32d86ce3,
0x45df5c75, 0xdcd60dcf, 0xabd13d59, 0x26d930ac, 0x51de003a,
0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599,
0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190,
0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f,
0x9fbfe4a5, 0xe8b8d433, 0x7807c9a2, 0x0f00f934, 0x9609a88e,
0xe10e9818, 0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
0xb66b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c6e95ed,
0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3,
0xfbd44c65, 0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a,
0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5,
0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa, 0xbe0b1010,
0xc90c2086, 0x5e68b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17,
0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6,
0x03b6e20c, 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615,
0x73dc1683, 0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1, 0xf00f9344,
0x8706d93d, 0x1e01f268, 0x6906c2fe, 0xf62575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a,
0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0x1bb67f1,
0xa6bc5767, 0x3fb506dd, 0x48b2364b, 0xd80d2bda, 0xaf0a1b4c,
0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef,
0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe,
0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31,
0x2cd99e8b, 0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c,
0x026d930a, 0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b,
0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1,
0x18b74777, 0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45, 0xa00ae278,
0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7,
0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66,
0x37d83bf0, 0xa9bcae53, 0xdeb9ec5, 0x47b2cf7f, 0x30b5ffe9,
0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605,
0xcdd70693, 0x54de7729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8,
0x5d681b02, 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b,
0x2d02ef8d
};
uint32_t sa_crc32(unsigned int seed, const unsigned char *buf, unsigned int len)
{

```




```
unsigned int crc32 = seed;
unsigned int i = 0;
for ( i = 0; i < len; i++)
{
    crc32 = crc32table_le[((crc32 & 0xff) ^ buf[i]) & 0xff] ^ (crc32 >> 8);
}
return (~crc32);
}
//-----
//2、报文payload计算CRC方法
//1) 对下发给SA的完整TCP报文数据不要padding数据，seed设置为0x0，直接调用
sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。最后把返回值赋值给
BD的pkt_crc16域段。
//2) 把返回值赋值给BD的pkt_crc16域段。
//-----
/* Polynomial num is 1021
const uint16_t crc16table_le[256] =
{
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedaе, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaа, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};
#define ALIGN_SIZE 32
uint16_t sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)
{
    uint16_t crc16 = seed;
    unsigned int i = 0;
    unsigned int alg_len = 0;
    alg_len = (len / ALIGN_SIZE) * ALIGN_SIZE + (((len % ALIGN_SIZE) == 0) ? 0 : ALIGN_SIZE);
    for ( i = 0; i < alg_len; i++)
    {
```




```

if (i < len) {
    crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ buf[i]) & 0xff] ^ (crc16 << 8);
} else {
    crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ 0x0) & 0xff] ^ (crc16 << 8);
}
}
return crc16;
}
//-----
//3、BD计算CRC方法
//1) 对下发给SA的64Byte的BD截取前面48Byte的数据，seed设置为0x0，直接调用
sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。最后把返回值赋值给BD的
pkt_crc16域段。
//2) 把返回值的高8bit赋值给BD的pkt_crc16域段的低8bit，把返回值的低8bit赋值给BD的pkt_crc16
域段的高8bit。
//-----
uint16_t sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)
{
    uint16_t crc16 = seed;
    unsigned int i = 0;
    for (i = 0; i < len; i++)
    {
        crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ buf[i]) & 0xff] ^ (crc16 << 8);
    }
    return crc16;
}

```

1. 配表数据计算CRC方法

配表数据进行32B对齐，padding数据填0x0。

seed设置为0xFFFFFFFF，调用sa_crc32(unsigned int seed, const unsigned char *buf, unsigned int len)进行计算；有多个2KB时，再次调用sa_crc32()时，seed值为上次CRC的32bit返回值按位取反。

2. 报文payload计算CRC方法

对下发给SA的完整TCP报文数据不要padding数据，seed设置为0x0，直接调用sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。

把返回值赋值给BD的pkt_crc16域段。

3. BD计算CRC方法

对下发给SA的64Byte的BD截取前面46Byte的数据，seed设置为0x0，直接调用sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。

把返回值的高8bit赋值给BD的bd_crc16域段的低8bit，把返回值的低8bit赋值给BD的bd_crc16域段的高8bit。

为了将部分硬件匹配引擎处理错误的信息送给软件，通过ADD_PACKET数据结构的User_Data（word5中）上报这些信息：

User_Data[3:0]：为0x0，表示BD执行正确；bit[0]为1，表示硬件引擎收到的任务存在错误；bit[1]为1，表示该匹配任务超时，只对同步任务生效；bit[2]为1，表示POE所携带的地址非法。

5.6.2 大小端转换

针对报文、描述信息、上报结果、知识库表项等不同数据，大小端格式固定如表5-6所示。



表 5-6 各数据接口大小端模式

数据	大小端	备注
BD	小端	送给SA的BD，软件新申请的系统内存，然后驱动会将整个内存拷贝，组成一个完整的BD下发给硬件SA加速器。如果产品软件有大端需求，后续由驱动进行大小端转换。
报文	大端	网络字节序，先收到的字符放在内存低地址。
知识库	小端	当前仅支持小端，如果后续CPU为大端，则由SA转换器进行适配。
匹配结果	小端	如果产品软件有大端需求，后续由驱动进行大小端转换。
寄存器访问(AXI Slave)	小端	当前仅支持小端。

5.6.3 知识库加载和回读接口

知识库表项加载和回读跟匹配任务一样走POE接口，通过BD+表项存储buf的方式交互。

详细流程描述参见“[知识库加载](#)”章节。

5.7 应用配置

请参见“[业务开工配置](#)”章节，该项配置流程下所有项均可动态更改。

知识库更新流程参考“[知识库加载](#)”章节。

5.8 初始化

5.8.1 知识库加载和回读接口

SA加速器初始化分为四步，硬件引擎全局配置----知识库编译----知识库加载----业务开工配置，必须按顺序执行。

如果发生芯片复位或者SA加速器单独复位，必须重新执行以上全流程；如果是仅软件复位/驱动卸载等，只需执行后三步流程。

5.8.2 硬件引擎全局配置

硬件引擎全局配置流程如下：

先根据[软复位](#)的[步骤5](#)描述的流程对SA进行复位、解复位操作，再进行下面的配置流程。

步骤1 初始化SA内部RAM--对REG_SA_INIT_CMD寄存器的bit[0]写1，使能SA内部RAM的初始化，100μs后，读取REG_SA_INIT_CMD寄存器，该寄存器bit[0]若为0时表示SA内部



RAM初始化完成，若为1则还在初始化，等待100μs后再次读取，若还为1则表示初始化失败。

步骤2 按芯片约定配置SA节点编号寄存器REG_PPM_SA_NODE_CODE_CFG。

步骤3 配置AXI接口相关寄存器--配置AXI接口相关寄存器REG_SA_BD_AXI_USER_CFG0~REG_SA_AXI_COMMON_CFG。

步骤4 配置POE任务获取地址寄存器--（即从POE读取GET_POE的地址，REG_PPM_GET_POE_ADDR_L32B_CFG, REG_PPM_GET_POE_ADDR_H16B_CFG）。

步骤5 配置POE任务反馈地址寄存器--（即向POE发送ADD_PACKET的地址，REG_PPM_ADD_POE_ADDR_L32B_CFG, REG_PPM_ADD_POE_ADDR_H16B_CFG）。

步骤6 配置队列号到优先级映射寄存器--据业务需要配置队列优先级寄存器REG_PPM_QID_PRI_CFG，建立Qid到高低优先级的映射。（注意要跟POE内SA相关队列的优先级配置吻合，PPM模块队列优先级配置，bit[1:0]对应q0，bit[3:2]对应q1，依次类推；每个队列的2bit优先级配置值的含义：0-低优先级业务队列，1-高优先级业务队列，2-表项下发队列）

步骤7 使能异常中断上报—如下寄存器内各个域都配置为1：REG_SA_BASE_INT_EN、REG_PPM_FIFO0_INT_EN、REG_PPM_FIFO1_INT_EN、REG_PPM_1B_ECC_INT_EN、RZEG_PPM_2B_ECC_INT_EN、REG_PPM_PROC_INT_EN、REG_PPM_ERR_BD_INT_EN、REG_CPM_FIFO1_INT_EN ~ REG_CPM_FIFO4_INT_EN、REG_CPM_1B_ECC_INT_EN、REG_CPM_2B_ECC_INT_EN、REG_CPM_PROC_INT_EN、REG_PMU_CTRL_FIFO0_FL_ERR_INT_EN ~ REG_PMU_CTRL_FIFO2_FL_ERR_INT_EN、REG_PMU_CTRL_1BIT_ECC_ERR_INT_EN、REG_PMU_CTRL_2BIT_ECC_ERR_INT_EN、REG_PMU_CTRL_ABNORMAL_ERR_INT_EN、REG_PMU_STR_FIFO_FL_ERR_INT_EN、REG_PMU_STR_ABNORMAL_ERR_INT_EN、REG_PMU_DFA_FIFO_FL_ERR_INT_EN、REG_PMU_DFA_ERR_INT_EN、REG_PMU_NFA_FIFO_FL_ERR_INT_EN、REG_PMU_NFA_FIFO_1BIT_ECC_ERR_INT_EN、REG_PMU_NFA_FIFO_2BIT_ECC_ERR_INT_EN、REG_PMU_NFA_ERR_INT_EN、REG_PMU_NFA_WARN_INT_EN、REG_BMF_HASH_FIFO_INT_EN、REG_BMF_PRO_INT_EN、REG_BMF_BADKEY_INT_EN、REG_BMF_COL_34B_FIFO_INT_EN、REG_BMF_6B_COL_FIFO_INT_EN、REG_BMF_COL_SEND_FIFO_INT_EN、REG_BMF_PRO_1BIT_ECC_INT_EN、REG_BMF_PRO_2BIT_ECC_INT_EN、REG_BMF_BADKEY_1BIT_ECC_INT_EN、REG_BMF_BADKEY_2BIT_ECC_INT_EN、REG_BMF_COL_1BIT_ECC_INT_EN、REG_BMF_COL_2BIT_ECC_INT_EN、REG_RPM_FIFO_FL_ERR_INT_EN、REG_RPM_1BIT_ECC_ERR_INT_EN、REG_RPM_2BIT_ECC_ERR_INT_EN、REG_RPM_ERR_INT_EN。

步骤8 使能动态时钟门控--寄存器REG_SA_CLK_GATE_EN_CFG各个域写1。

步骤9 使能ECC 1bit纠错回写--对REG_BMF_ECC_CORRECT_EN_CFG[0]、REG_PMU_ECC_1BIT_ERR_WB_EN[0]写1。

步骤10 各用户接口访问AXI的OTD门限—用默认值可正常工作，不需配置；默认值不合理时，才需在此处做额外配置。

步骤11 相关寄存器：REG_PPM_RD_BD_OTD_NUM_CFG、REG_PPM_RD_BUF_INFO_OTD_NUM_CFG、REG_PPM_RD_PKT_OTD_NUM_CFG、REG_PPM_WR_TB_DATA_OTD_NUM_CFG、REG_CPM_RD_FLLK_OTD_NUM_CFG、



REG_CPM_WR_FLLK_OTD_NUM_CFG、REG_BMF_RD_LINK_OTD_NUM_CFG、REG_PMU_CACHE_REN_INST_OTD_NUM_CFG和REG_RPM_AXI_OTD_NUM_CFG，主要根据业务场景下描述符、报文、指令、结果等的流量比例设置。

步骤12 匹配超时中止设置--用默认值可正常工作，不需配置；默认值不合理时，才需在此处做额外配置。

相关寄存器：REG_PMU_NFA_VEC_SYNC_TIMEOUT_CFG、REG_PMU_NFA_TCP_SYNC_TIMEOUT_CFG、REG_PMU_NFA_VEC_ASYNC_TIMEOUT_CFG、REG_PMU_NFA_TCP_ASYNC_TIMEOUT_CFG，典型场景下单个报文匹配处理时间为10μs级别，由软件根据业务场景下对性能和匹配结果数量进行取舍来配置。

步骤13 hash伪命中数量门限设置—用默认值可正常工作，不需配置；默认值不合理时，才需在此处做额外配置。

相关寄存器：REG_BMF_SYNC_MAX_6B_HASH_HIT_CFG、REG_BMF_SYNC_MAX_4B_HASH_HIT_CFG、REG_BMF_SYNC_MAX_3B_HASH_HIT_CFG、REG_BMF_MAX_ASYNC_6B_HASH_HIT_CFG、REG_BMF_MAX_ASYNC_4B_HASH_HIT_CFG、REG_BMF_MAX_ASYNC_3B_HASH_HIT_CFG，为保证极端场景下匹配命中率，上述三组寄存器默认值均为0x1FFF（保留8K个伪命中），由软件根据业务场景下对性能和匹配结果数量进行取舍来调整。

如果发生芯片复位或者SA加速器单独复位，必须重新执行以上全流程；如果是仅软件复位/驱动卸载等，只需执行后三步流程。

----结束

5.8.3 知识库编译

请参见“[应用配置](#)”章节。

5.8.4 知识库加载

5.8.4.1 知识库编译结果

SA转换器将输入模式串转换成硬件引擎能识别的知识库表项，结构体形式参照“[编译结果结构体](#)”章节。

须知

因为产品对编译结果输出要求为上述结构体形式，所以转换器的输出不以结果文件形式存在，测试时需要测试平台对上述编译结果进行保存，否则每次都要执行编译。EMU和FPGA原型由于CPU核的时钟频率非常低，所以只能通过离线编译后传输知识库文件到设备上的方法来测试。

5.8.4.2 加载流程

SA转换器将输入模式串转换成硬件引擎能识别的知识库表项，结构体形式参照“[编译结果结构体](#)”章节。



表 5-7 知识库表项分组

表项分组	Group1	Group2	Group3	Group4
表项类型 (data_type, 十进制)	6/7/12/14/15/16/17/18	9/10/11	5	4 (此为分库指令表, 分多 段, 分别对应 多个lib)

Group3为BMF片外Linklist，占用内存大小为128K*8Byte*n（n为软件调接口时配置的宽度，取值16/32/64，默认是16）。

Group4为PMU片外指令表，占用内存最大为 m*1.3*2kByte（m为模式串数量）。



图 5-4 SA 硬件知识库加载总体流程图

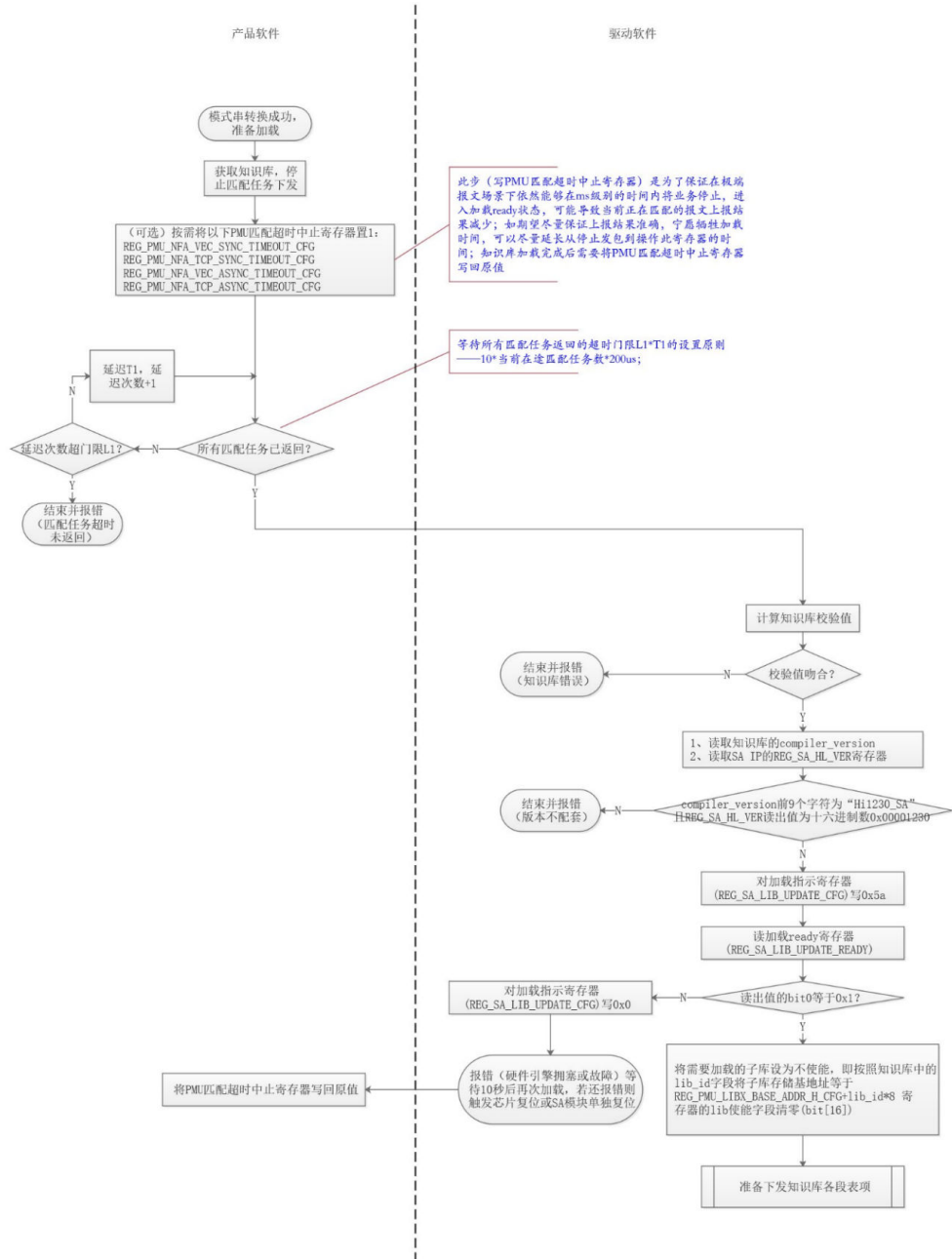




图 5-5 编译结果单数据段写入流程图

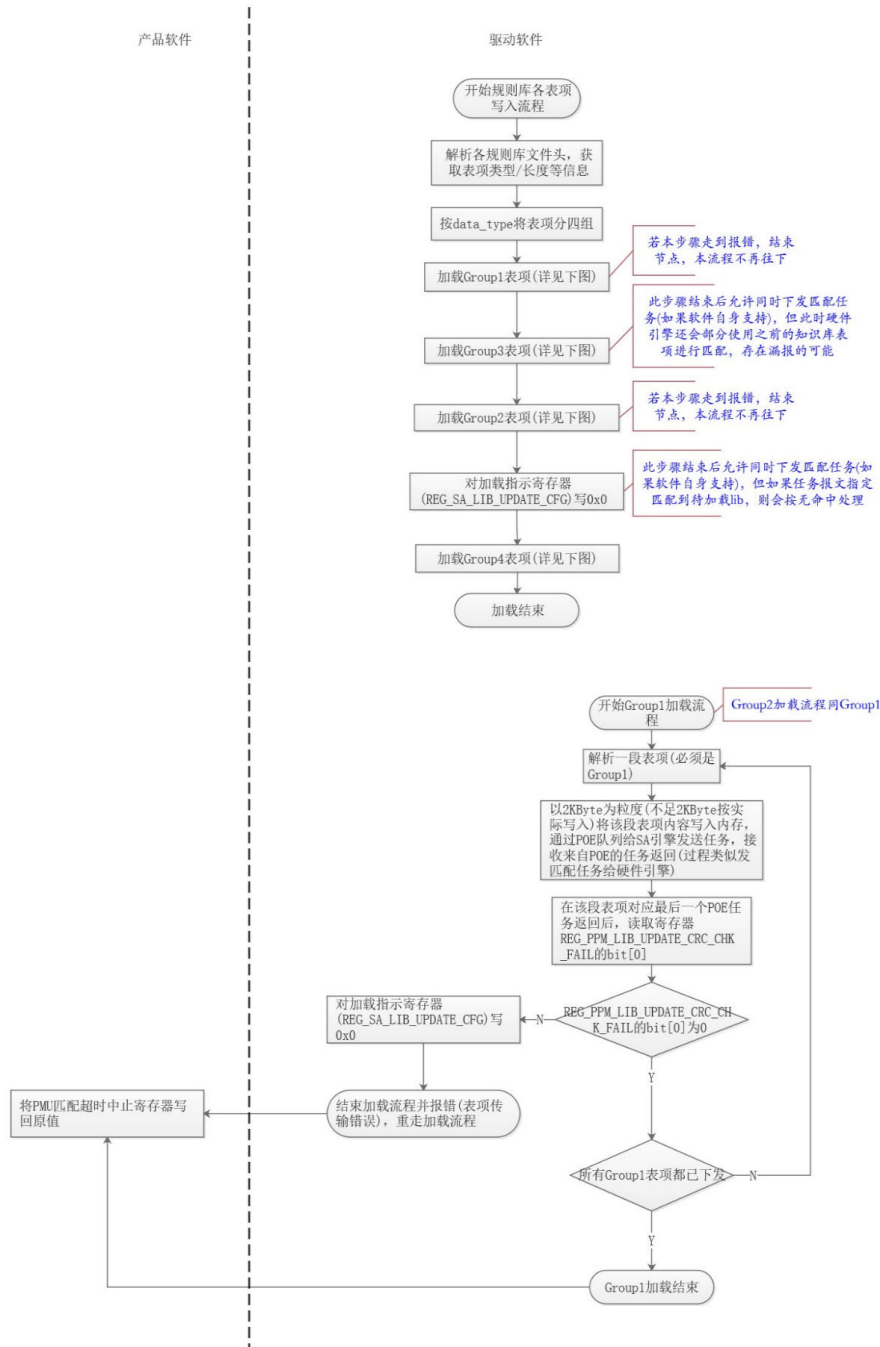
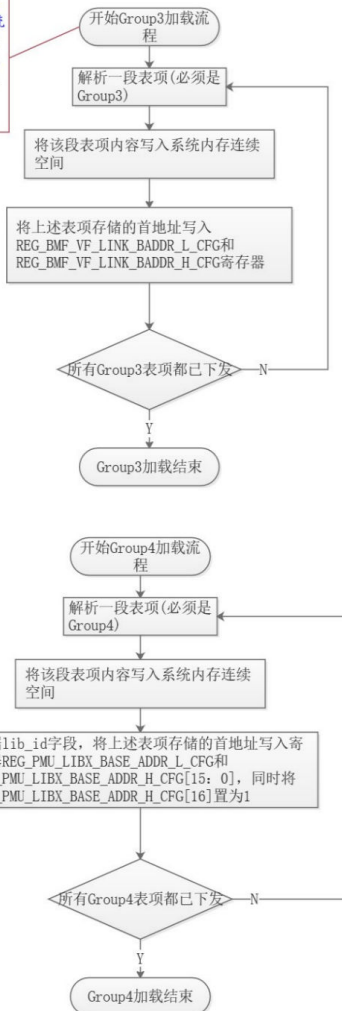


图 5-6 加载流程

产品软件

驱动软件

如果系统内存足够大，可以提前将本组表项(十几MB到两百多MB不等)写入系统内存(非升级前该表项存储区域)，在此流程中只需要将新存储首地址和使能位写入硬件寄存器，这样能减少不能正常执行匹配任务的时间；Group4表项于此类似；



5.8.5 业务开工配置

配置流程如下：

- 步骤1** 配置流表存放首地址寄存器（REG_CPM_FLOW_BASE_ADDR_L_CFG，REG_CPM_FLOW_BASE_ADDR_H_CFG），该地址对应软件内存中划分硬件引擎专用的流表存储区域首地址，此组寄存器与流数量（xM条流）对应。
- 步骤2** 业务开工--对REG_SA_EN_CFG写入0x55，指示硬件引擎进入开工状态；如果要停工，则需写入0xAA。

----结束



5.8.6 FE 中断 AXI 总线关闭后打开流程

SA硬件引擎出现ecc 2bit等FE中断时，如果SA BASE_PF_REG的REG_SA_DBW_SHUT_DOWN_EN配置为1，会将SA AXI总线收发关闭，以避免故障进一步扩散。如不需此功能，请将REG_SA_DBW_SHUT_DOWN_EN配置为0。

关闭后再次打开流程如下：

步骤1 对SA BASE_PF_REG的REG_SA_DBW_SHUT_DOWN_EN寄存器写0，AXI总线收发即可恢复。

步骤2 如需再打开FE总线关闭功能，需要将此寄存器写1。

----结束

5.8.7 软复位

软件需要按照下述流程对SA进行软复位准备，各步骤都要顺序执行：

步骤1 软件确保业务数据流停止，不再下发业务；

步骤2 关闭总线接口：

1. 下发关闭总线接口请求：

a. 查询《Hi309a 寄存器手册》的寄存器偏移位的bit[0]（域ctrl_shutdown_req）配置为0x1，则不再发起新的总线命令。

2. 查询总线命令处理状态：

a. 查询《Hi309a 寄存器手册》的寄存器AM_CURR_TRANS_RETURN（偏移为）为0x3时，表征之前发出的所有总线操作都已经被Master接口处理完毕。

b. 如果软件在自己规定时间内（建议值不超过10ms）未查询到上述寄存器值变为0x3，这预示出现总线死锁，复位准备无法完成，建议记录各模块总线相关信息后复位整个芯片。

须知

发生此种情况预示出现总线死锁，推荐做法是记录各模块总线相关信息后复位整个芯片。

步骤3 软件确保POE模块中与SA绑定的队列都已被读空，若此时相应队列未非空状态，则软件需要将该队列读空。

步骤4 在复位SA前，根据需要收集寄存器（如中断、状态、统计）。

步骤5 软复位SA：

当SA的复位准备完成后，通过操作《Hi309a 寄存器手册》中有关SA的寄存器，来完成SA的复位操作：

1. 配置SC_SA_ICG_DIS（0x324）为0b111，关闭模块时钟。

2. 查询SC_SA_ICG_ST（0x5320），若查询值为0b000，则表示模块时钟关闭。

3. 配置SC_SA_RESET_REQ（0xA20）为0xF，对SA模块进行复位。



4. 查询SC_SA_RESET_ST (0x5A20) , 若查询值为0xF, 则表示模块处于复位状态。
5. 配置SC_SA_ICG_EN (0x320) 为0b111, 打开模块时钟。
6. 查询SC_SA_ICG_ST (0x5320) , 若查询值为0b111, 则表示模块时钟打开。
7. 等待500ns, 配置SC_SA_ICG_DIS (0x324) 为0b111, 关闭模块时钟。
8. 查询SC_SA_ICG_ST (0x5320) , 若查询值为0b000, 则表示模块时钟关闭。
9. 软件判断软复位已经完成, 执行以下步骤进行解复位。
10. 配置SC_SA_RESET_DREQ (0xA24) 为0xF, 对SA模块进行解复位。
11. 查询SC_SA_RESET_ST (0x5A20) , 若查询值为0xF, 则表示模块处于解复位状态。
12. 配置SC_SA_ICG_EN (0x320) 为0b111, 打开模块时钟。
13. 查询SC_SA_ICG_ST (0x5320) , 若查询值为0b111, 则表示模块时钟打开。

步骤6 软件读清SA寄存器状态:

SA计数类寄存器没有复位, 需要软件需要执行以下操作完成SA模块相关状态和统计寄存器清空功能。

1. 配置《Hi309a 寄存器手册》中BASE_PF_REG的REG_SA_CNT_CLR_CE. cnt_clr_ce 为1, 读清操作使能。
2. 软件执行读操作清空各模块寄存器页签中所有CNT_CYC和CNT类型寄存器。
3. 配置《Hi309a 寄存器手册》中BASE_PF_REG的REG_SA_CNT_CLR_CE. cnt_clr_ce 为0, 读清操作去使能。

步骤7 对SA执行初始化配置流程。

以下步骤对POE模块寄存器进行操作, POE驱动已封装接口, 请直接调用对应接口。

步骤8 把POE中SA对应的POE_HAC_ADD_CHN_CFG寄存器对应位配置为1 (需要先将寄存器POE_DEBUG_MODE_EN配置为0x5A5A) 。

步骤9 把POE中SA对应的POE_GET_OTD_SN_INIT_EN[15:8]和POE_GET_OTD_SN_INIT_EN[7:0]寄存器值写1, 初始化添加任务的SN号 (需要先将寄存器POE_DEBUG_MODE_EN配置为0x5A5A) 。

步骤10 把POE中SA对应的POE的POE_GET_OTD_SN_INIT_EN[31:16]寄存器值写1, 初始化获取任务的SN号 (需要先将寄存器POE_DEBUG_MODE_EN配置为0x5A5A) 。

须知

如果POE被复位, 再次启动使用之前, 需要对SA的REG_PPM_RST_POE_CREDIT_CFG寄存器写入0x3, 将SA内部的credit sn恢复为初始值0和将credit用量计数清0。

----结束

5.9 DFX 应用指导

5.9.1 故障在线检测



5.9.1.1 SA 配置异常

对SA描述符（BD）和buf_info信息进行合法性检查，异常处理如表5-8所示。

表 5-8 描述信息异常处理机制

异常类型	异常描述	内部处理机制	是否上报中断和记录异常地址	外部处理机制
buf_info_len_chk_err	pkt_rd读取buf类型报文时其长度总和和BD长度域不一致错误	按pkt_len或buf_info中的长度容错处理，不保证读取报文内容的正确性	上报中断，不记录地址	软件查找原因
pkt_crc_chk_err	报文内容CRC校验错误	不影响处理	上报中断	软件查找原因
bd_crc_chk_err	BD内容CRC校验错误	不影响处理	上报中断	软件查找原因
data_type_err	TB BD的data_type错误，不在使用的data_type范围内	BD直接返回CPU，不做处理	上报中断，不记录地址	软件查找原因
qm_type_err	POE的优先级0/1队列读出非业务类型报文（配表类型），优先级2队列读出业务类型报文（非配表类型）错误	BD直接返回CPU，不做处理	上报中断，不记录地址	软件查找原因
bd_type_err	BD的类型不是0、2~3	BD直接返回CPU，不做处理	上报中断，不记录地址	软件查找原因
buf_num0_err	BD中的link_flag标记为1，且buf_num的个数为0	BD直接返回CPU，不做处理	上报中断，不记录地址	软件查找原因
rslt_addr_check_err	BD中的结果地址检测不在设定的上下限范围内（只比高24bit，设比较开关）	BD直接返回CPU，不做处理	是	软件查找地址超范围原因
crs_addr_check_err	BD中的跨包地址检测不在设定的上下限范围内（只比高24bit，设比较开关）	BD直接返回CPU，不做处理	是	软件查找地址超范围原因



异常类型	异常描述	内部处理机制	是否上报中断和记录异常地址	外部处理机制
buf_info_addr_check_err	向量 (buf_info) 内地址检测不在设定的上下限范围内 (只比高24bit, 设比较开关)	正常处理 (会导致读异常地址)	是	软件查找地址超范围原因
pkt_addr_check_err	BD中的报文地址检测不在设定的上下限范围内 (只比高24bit, 设比较开关)	BD直接返回CPU, 不做处理	是	软件查找地址超范围原因
bd_len0_err	收到BD的pkt_len域为0	BD直接返回CPU, 不做处理	上报中断, 不记录地址	软件查找原因
sa_no_en_err	收到BD, 但SA未开工	BD直接返回CPU, 不做处理	上报中断, 不记录地址	软件查找原因
lib_update_err	lib update未拉高或加载ready未拉高时收到加载写BD	BD直接返回CPU, 不做处理	上报中断, 不记录地址	软件查找原因

5.9.1.2 典型问题定位

5.9.1.2.1 SA 加速器不获取任务不开工或无反应

- 检查SA开工配置寄存器REG_SA_EN_CFG (偏移地址: 0x40034) 是否为0x55, 写0x55表示开工。
- 检查SA初始化寄存器REG_SA_INIT_STATUS (偏移地址: 0x80B08) 是否为0xF, 如果该值不为0xF, 软件需确认是否已正确执行SA初始化过程。
- 检查PPM模块FIFO状态寄存器REG_PPM_POE_INTF_STATUS (偏移地址: 0x90B40) bit[0] (sa_task_vld) 是否为1, 1表示POE对SA输入的task有效。
- 检查PPM模块发起TASK读次数统计REG_PPM_TX_TASK_RD_CNT (偏移地址: 0x90E10)、PPM模块收到有效TASK个数统计REG_PPM_BD_RD_TX_BD_CNT (偏移地址: 0x90E0C), 若均有值说明读取POE TASK功能基本正常。
- 检查PPM模块收到无效TASK个数统计REG_PPM_RX_INVLD_TASK_CNT (偏移地址: 0x90E14), 若有值说明POE TASK无效, 需要POE模块进行定位分析。
- 检查各模块中断寄存器和数据通道的反压状态寄存器再有无中断, 再根据中断含义进一步定位。



PPM模块出现“报文地址检测不在设定的上下限范围内”或“向量内地址检测不在设定的上下限范围内”或“跨包地址检测不在设定的上下限范围内”或“结果地址检测不在设定的上下限范围内”中断。

- 说明已开启BD内或buf_info内报文地址上下限检测功能，请检查上下限配置寄存器是否正确配置，相关寄存器：
 - REG_PPM_PKT_ADDR_MIN（偏移地址：0x42040）
 - REG_PPM_PKT_ADDR_MAX（偏移地址：0x42044）
 - REG_PPM_BUF_INFO_ADDR_MIN（偏移地址：0x42048）
 - REG_PPM_BUF_INFO_ADDR_MAX（偏移地址：0x4204C）
 - REG_PPM_CRS_ADDR_MIN（偏移地址：0x42050）
 - REG_PPM_CRS_ADDR_MAX（偏移地址：0x42054）
 - REG_PPM_RSLT_ADDR_MIN（偏移地址：0x42058）
 - REG_PPM_RSLT_ADDR_MAX（偏移地址：0x4205C）
- 通过寄存器REG_PPM_CHK_ERR_ADDR_L32B（偏移地址：0x90B10）和REG_PPM_CHK_ERR_ADDR_H16B（偏移地址：0x90B14）查看出错的地址。
- 检查BD内报文（或buf_info的地址）、跨包、结果地址填写是否正确，或buf_info内报文地址是否填写正确。

5.9.1.2.2 PPM 模块出现“pkt_rd 读取 buf 类型报文时其长度总和和 BD 长度域不一致错误”

- 说明已使用多buf方式发包，请检查多buf长度之和是否等于BD报文长度域值。
- 检查buf_info内长度域填写是否正确，或BD的buf_info地址是否填写正确。

5.9.1.2.3 SA 出现“AXI 总线响应 resp 异常中断”

- 出现该中断一般是AXI访问地址非法导致，检查出现该中断的模块，进而确定是哪些类型地址非法。

表 5-9 AXI 总线响应 resp 异常中断

出现AXI总线响应resp异常中断模块	可能的地址非法	伴随信息寄存器
PPM出现AXI总线响应resp异常（非OKAY）中断。	<div>1. TASK读取地址非法，检查REG_PPM_GET_POE_ADDR_L32B_CFG（0x90030）、REG_PPM_GET_POE_ADDR_H16B_CFG（0x90034）配置是否正确。</div> <div>2. BD读取地址非法，检查TASK内的BD地址是否填写正确。</div> <div>3. 读取报文、buf_info地址非法，检查BD内的地址是否填写正确。</div>	通过寄存器REG_PPM_AXI_ERR_STATU S（0x90B24）的axi_resp_err_loca确定是哪 种操作引起中断，通过axi_err_resp确定resp具体的异常。



出现AXI总线响应resp异常中断模块	可能的地址非法	伴随信息寄存器
CPM读取流表/跨包数据时，AXI总线响应rresp异常（非OKAY）中断；流表、跨包数据写入时，AXI总线响应bresp异常（非OKAY）中断。	1. 流表地址非法，检查REG_CPM_FLOW_BASE_ADDR_L_CFG（0x43000+flow_num*8）、REG_CPM_FLOW_BASE_ADDR_H_CFG（0x43004+flow_num*8）配置是否正确。 2. 跨包地址非法，检查BD内的跨包地址是否填写正确。	通过寄存器REG_CPM_AXI_ERR_RESP_STATUS（0x98B10）查看错误信息。
BMF sa_axi_rd模块调度超时错误中断。	linklist地址非法，检查REG_BMF_VF_LINK_BADDR_L_CFG（0x44000）、REG_BMF_VF_LINK_BADDR_H_CFG（0x44004）配置是否正确。	通过寄存器REG_BMF_COL_AXI_STATUS（0xA0B2C）的bmf_axi_ll_err_rresp字段查看错误信息。
PMU AXI读返回resp非零错误。	指令访问地址非法，检查REG_PMU_LIBX_BASE_ADDR_L_CFG（0x45000+lib_id*8）、REG_PMU_LIBX_BASE_ADDR_H_CFG（0x45004+lib_id*8）配置是否正确。	通过寄存器REG_PMU_CACHE_AXI_ERR_RRESP（0xA8B1C）查看错误信息。
RPM AXI写返回resp非零错误。	结果地址非法，检查BD内的结果地址是否填写正确。	通过寄存器REG_RPM_AXI_ERR_RESP（0xC8B54）查看错误信息。

5.9.1.2.4 SA 出现下发匹配任务后不返回响应消息（ADD_PACKET）

- 出现这种情况，一般是软件构造的POE消息或BD内容有误，或外存表项的基址配置有误，导致SA发出错误的AXI访问地址，且系统未配置DAW窗口，SMMU模块收到错误地址访问后挂死。查看Master DBW的AM_CURR_TRANS_RETURN（偏移地址：0x88150），确定是否有未完成的AXI读写操作。
- 如第一步发现有未完成的AXI读写操作，可以通过SMMU的DFX进一步找出错误地址，同时排查POE消息、BD内容及外存表项的基址配置是否有误，或是否存在内存踩踏导致BD内容被改写的情况。

5.9.1.2.5 SA 出现 BD 内容 CRC 校验错误或报文内容 CRC 校验错误

- 出现这种情况，一般是开启了BD和报文的CRC校验功能，但是BD中没有正确填写BD和报文的CRC校验值。
- 检查BD中BD和报文的CRC校验值是否填写正确。
- 检查是否存在内存踩踏导致BD和报文内容被改写的情况。
- 如不希望使用该功能，通过对寄存器REG_PPM_CRC_CHK_EN（偏移地址：0x90060）对应bit写0进行关闭。



5.9.1.2.6 SA 出现知识库加载超时中断

- 检查寄存器REG_SA_LIB_UPDATE_TIMEOUT_CFG（偏移地址：0x8010C）配置门限是否过小。
- 如超时门限配置无问题，检查软件处理过程是否存在耗时过长的情况。

5.9.1.2.7 CPM 模块出现流表基地址无效异常中断

- 检查软件流号使用的范围。
- 检查流表基地址配置寄存器：REG_CPM_FLOW_BASE_ADDR_H_CFG（偏移地址：0x43004+flow_num*8），流号范围内的基地址是否已经正确配置，flow_base_vld是否置1。

5.9.1.2.8 CPM 模块出现流表数据奇偶校验异常中断

- 检查软件是否在新流的第一个BD中打上新流标记。
- 读取统计寄存器：REG_CPM_PROC_NEW_FLAG_CNT，检查和软件下发的流数量是否一致。
- 读取统计寄存器：REG_CPM_SND_BMF_PKT_CNT，检查和软件下发的任务数量是否一致，不一致则说明有任务丢失，检查前级路径统计。
- 检查流表内存地址分配，是否和其他内存有冲突。

5.9.1.2.9 CPM 模块出现流表队列一致性异常中断

- 检查软件同流队列号使用是否一致。
- 读取统计寄存器：REG_CPM_PROC_NEW_FLAG_CNT，检查和软件下发的流数量是否一致。

5.9.1.2.10 分组匹配结果有丢失现象一

步骤1 检查BMF分组hash假命中超限中止配置寄存器：

- REG_BMF_SYNC_MAX_6B_HASH_HIT_CFG（偏移地址：0xA0000）。
- REG_BMF_SYNC_MAX_4B_HASH_HIT_CFG（偏移地址：0xA0004）。
- REG_BMF_SYNC_MAX_3B_HASH_HIT_CFG（偏移地址：0xA0008）。
- REG_BMF_MAX_ASYNC_6B_HASH_HIT_CFG（偏移地址：0xA000C）。
- REG_BMF_MAX_ASYNC_4B_HASH_HIT_CFG（偏移地址：0xA0010）。
- REG_BMF_MAX_ASYNC_3B_HASH_HIT_CFG（偏移地址：0xA0014）。

上述6个寄存器的bit[16]是否为1，若任意一个为1，进入**步骤2**；若均为0，非本章节描述故障。

步骤2 检查 REG_BMF_3B_DROP_CNT（偏移地址：0xA0E3C）、REG_BMF_4B_DROP_CNT（偏移地址：0xA0E40）、REG_BMF_6B_DROP_CNT（偏移地址：0xA0E44）任意寄存器的值是否不为0，若不为0，则判断为分组匹配结果过多而被丢弃。

----结束

解决方案：

1. 可将**步骤1**所述寄存器bit[16]均置0，则关闭hash假命中超限功能。
2. 可将**步骤1**所述寄存器的bit[15:0]设置为比现有值更大的值，例如0x3FFF或更大值。



5.9.1.2.11 分组匹配结果有丢失现象二

- 检查BD中是否开启“samecut”功能，BD中samecut=1为打开“samecut”功能。
- 检查报文的内容是否为连续重复字节，比如报文为100{a}。
- 若key值为aaa，则只上报前8个和后8个分组结果，对某些场景可能导致部分命中结果丢失。解决方案：可关闭samecut功能。

5.9.1.2.12 分组匹配结果有丢失现象三

- 检查寄存器REG_BMF_PRO_RINT [2]，查看是否有4个libid均无效的告警。
- 若有，则可能导致无法分组命中，检查软件下发的子库ID的合法性。

5.9.1.2.13 无分组命中结果现象

- 检查是否有包长小于3Byte的数据包。
- 检查跨包使能开关是否打开（BD中的cross_en域段），跨包使能不使能则无法匹配命中。

5.9.1.2.14 匹配结果有丢失现象

- 检查RPM模块匹配结果达到允许发送的最大个数后丢弃的匹配结果个数统计寄存器：（REG_RPM_SEND_MRESULT_RMAX_DROP_CNT）和匹配结果头中的cut_flag。
- 当去重后的匹配结果数达到允许上报的最大个数后，该TCP报文中的其余匹配结果均被丢弃。

5.9.1.2.15 匹配结果中有重复的命中项

- 检查寄存器REG_RPM_MATCH_DIS_CNT（匹配结果遇到hash冲突且超过逃生桶深的匹配结果个数统计）和匹配结果头中的err_info[3]，若有值则表示曾经发生去重失败的情况，重复的命中信息由其引起；否则可能是输出匹配结果的模块有bug或数据在总线/内存中出错。

5.9.1.2.16 所有报文均上报无命中的匹配结果（预期有命中）

- 先按无分组命中分析。
- 读取lib_offset表项（Type=15），判断报文所指定匹配子库在表项中的VLD是否无效，必要时与编译后的规则库文件中的对应表项比对。
- 读取lib_base表项（首地址0x45000），判断报文所指定匹配子库在表项的内容是否跟对应子库指令表在内存中的物理地址吻合。
- 若以上无发现，则需深入分析各模块中断及其他DFX寄存器、报文和规则。

5.9.1.2.17 匹配结果中缺失部分预期的命中项

- 先按照分组匹配结果有都是现象、无分组命中结果现象、匹配结果有丢失现象分析。
- 读取匹配结果头中的err_info字段，有几项可能导致缺失命中项的error描述。
- SA开发人员根据已有匹配结果和报文激励（如果能获取到）分析预期命中项是否合理，是否存在相似的命中项等等，深入分析分组过滤和精确匹配模块的中断以及各级状态和统计。



5.9.2 模块维测

5.9.2.1 内部状态查询

表 5-10 SA 内部 fifio 状态寄存器

状态名称	状态寄存器名	偏移地址
SA初始化状态指示	REG_SA_INIT_STATUS	0x80B08
SA加载规则库READY寄存器	REG_SA_LIB_UPDATE_READY	0x40B04
PPM模块FIFO0状态寄存器	REG_PPM_FIFO0_STATUS	0x90B00
PPM模块FIFO1状态寄存器	REG_PPM_FIFO1_STATUS	0x90B04
PPM模块地址检测错误对应的地址低32位	REG_PPM_CHK_ERR_ADDR_L32B	0x90B10
PPM模块地址检测错误对应的地址高16位	REG_PPM_CHK_ERR_ADDR_H16B	0x90B14
PPM模块缓存控制递增递减错误状态	REG_PPM_BUF_CTRL_ERR_STATUS	0x90B20
PPM模块AXI访问错误状态	REG_PPM_AXI_ERR_STATUS	0x90B24
PPM模块与POE接口相关状态寄存器	REG_PPM_POE_INTF_STATUS	0x90B40
PPM模块加载规则库CRC校验失败指示寄存器	REG_PPM_LIB_UPDATE_CRC_CHK_FAIL	0x42A00
CPM模块FIFO1状态寄存器	REG_CPM_FIFO1_STATUS	0x98B00
CPM模块FIFO2状态寄存器	REG_CPM_FIFO2_STATUS	0x98B04
CPM模块FIFO3状态寄存器	REG_CPM_FIFO3_STATUS	0x98B08
CPM模块FIFO4状态寄存器	REG_CPM_FIFO4_STATUS	0x98B0C
CPM AXI接口错误响应状态	REG_CPM_AXI_ERR_RESP_STATUS	0x98B10
CPM 两路队列的同流阻塞检测状态	REG_CPM_FT_CHK_BUSY_STATUS	0x98B14



状态名称	状态寄存器名	偏移地址
CPM FLOW_RD CACHE bitmap	REG_CPM_CAH_BITMAP0_STAT US	0x98B 18
CPM PKT_TRANS FSM历史状态低32 位	REG_CPM_TR_FSM_LOCK_L_STA TUS	0x98B 24
CPM PKT_TRANS FSM历史状态高32 位	REG_CPM_TR_FSM_LOCK_H_STA TUS	0x98B 28
CPM FLLK_WR FSM历史状态	REG_CPM_FK_FSM_LOCK_STATU S	0x98B 2C
CPM FLLK流表回写通道状态	REG_CPM_FT_WB_STATUS	0x98B 30
BMF模块3B hash过滤第一级结果缓存 fifo状态	REG_BMF_HASH_3B_LV1_FIFO_S TATUS	0xA0B 00
BMF模块BAD_KEY fifo状态	REG_BMF_BAD_KEY_STATUS	0xA0B 24
BMF模块BMF_COL模块 fifo状态	REG_BMF_COL_FIFO_STATUS	0xA0B 28
BMF模块BMF_COL AXI_RD模块 状态	REG_BMF_COL_AXI_STATUS	0xA0B 2C
BMF模块BMF_PRO fifo状态	REG_BMF_PRO_FIFO_STATUS	0xA0B 30
CTRL模块fifo状态0	REG_PMU_CTRL_FIFO0_STATUS	0xA8B 00
CTRL模块fifo状态1	REG_PMU_CTRL_FIFO1_STATUS	0xA8B 04
CTRL模块fifo状态2	REG_PMU_CTRL_FIFO2_STATUS	0xA8B 08
cache的buf0中lru状态	REG_PMU_CACHE_BUF0_STATU S	0xA8B 10
cache的2bit ecc错误地址锁存	REG_PMU_CACHE_2BIT_ECC_AD DR	0xA8B 18
AXI读返回resp错误锁存值	REG_PMU_CACHE_AXI_ERR_RRE SP	0xA8B 1C
ctrl的运行状态	REG_PMU_CTRL_PROC_STATUS	0xA8B 54
ctrl模块addr_fifo存放地址数量	REG_PMU_CTRLADDR_FIFO_NU M	0xA8B 58



状态名称	状态寄存器名	偏移地址
片外str的lru16权重最大索引	REG_PMU_CACHE_BUF0_INDEX_2_MAX	0xA8B70
片外str的lru16权重最大值	REG_PMU_CACHE_BUF0_WIGHT_2_MAX	0xA8B74
片外str的lru4权重最大索引	REG_PMU_CACHE_BUF0_INDEX_3_MAX	0xA8B78
片外str的lru4权重最大值	REG_PMU_CACHE_BUF0_WIGHT_3_MAX	0xA8B7C
字符串引擎的fifo状态	REG_PMU_FILTER0_FIFO_STATU S	0xB0B00
filter的运行状态	REG_PMU_FILTER_STATUS	0xB0B10
str的运行状态	REG_PMU_STR_STATUS	0xB0B18
str的最后一个head域	REG_PMU_STR_LAST_CFG	0xB0B20
dfa引擎的fifo状态	REG_PMU_DFA_FIFO_STATUS	0xB8B00
dfa引擎0保存状态机的最近五个状态	REG_PMU_DFA0_FSM_STATUS	0xB8B10
dfa引擎0保存的最后一个指令低32位	REG_PMU_DFA0_LAST_INS_L32 B	0xB8B14
dfa引擎0保存的最后一个指令高32位	REG_PMU_DFA0_LAST_INS_H32 B	0xB8B18
dfa引擎0扫描的字节数	REG_PMU_DFA0_MOVE_NUM	0xB8B1C
dfa0的指令错误类型	REG_PMU_DFA0_NODE_ERR_TY PE	0xB8B30
nfa的fifo状态	REG_PMU_NFA_FIFO_STATUS	0xC0B00
nfa0的状态1	REG_PMU_NFA0_1STATUS	0xC0B10
nfa0的状态2	REG_PMU_NFA0_2STATUS	0xC0B14
nfa0的状态3	REG_PMU_NFA0_3STATUS	0xC0B18



状态名称	状态寄存器名	偏移地址
nfa0的状态4	REG_PMU_NFA0_4STATUS	0xC0B1C
nfa0的状态5	REG_PMU_NFA0_5STATUS	0xC0B20
nfa0的匹配耗时最长vec对应的的分组信息	REG_PMU_NFA0_LONGTIME_GROUP_ID	0xC0B24
nfa0最后第1个指令	REG_PMU_NFA0_LAST_1INS	0xC0B28
nfa0最后第2个指令	REG_PMU_NFA0_LAST_2INS	0xC0B2C
nfa0最后第3个指令	REG_PMU_NFA0_LAST_3INS	0xC0B30
nfa0最后第4个指令	REG_PMU_NFA0_LAST_4INS	0xC0B34
nfa0最后第5个指令	REG_PMU_NFA0_LAST_5INS	0xC0B38
nfa0最后第6个指令	REG_PMU_NFA0_LAST_6INS	0xC0B3C
nfa0最后第7个指令	REG_PMU_NFA0_LAST_7INS	0xC0B40
nfa0最后第8个指令	REG_PMU_NFA0_LAST_8INS	0xC0B44
nfa0的指令错误类型	REG_PMU_NFA0_NODE_ERR_TYPE	0xC0B50
RPM模块FIFO状态	REG_RPM_FIFO_STATUS	0xC8B00
SA处理指定lib_id的最大延时	REG_RPM_CHK_LIB_ID_MAX_DELAY	0xC8B10
SA处理指定lib_id的最小延时	REG_RPM_CHK_LIB_ID_MIN_DELAY	0xC8B14
SA处理指定lib_id的总延时	REG_RPM_CHK_LIB_ID_ALL_DELAY	0xC8B18
SA处理同步task的最大延时	REG_RPM_CHK_SYN_MAX_DELAY	0xC8B1C
SA处理同步task的最小延时	REG_RPM_CHK_SYN_MIN_DELAY	0xC8B20



状态名称	状态寄存器名	偏移地址
SA处理同步task的总延时	REG_RPM_CHK_SYN_ALL_DELAY	0xC8B24
SA处理异步task的最大延时	REG_RPM_CHK_ASYNC_MAX_DELAY	0xC8B28
SA处理异步task的最小延时	REG_RPM_CHK_ASYNC_MIN_DELAY	0xC8B2C
SA处理异步task的总延时	REG_RPM_CHK_ASYNC_ALL_DELAY	0xC8B30
SA处理指定lib_id的任务数	REG_RPM_CHK_LIB_ID_TCP_CNT	0xC8B40
SA处理同步task统计	REG_RPM_CHK_SYN_TCP_CNT	0xC8B44
SA处理异步task统计	REG_RPM_CHK_ASYNC_TCP_CNT	0xC8B48
RPM模块pkt_sn使用状态	REG_RPM_PKT_SN_STATUS	0xC8B4C
RPM模块接收到CPM脉冲数与释放数的差值	REG_RPM_CPM_CNT_STATUS	0xC8B50
RPM模块AXI错误锁存	REG_RPM_AXI_ERR_RESP	0xC8B54
RPM模块TASK 超时配置值	REG_RPM_TASK_TIMEOUT_VALUE	0x49B00

5.9.2.2 统计

5.9.2.2.1 功能统计

表 5-11 功能统计寄存器

统计名称	统计寄存器名	偏移地址
SA统计读清使能（全局有效，写0：统计不读清；写1：统计读清）	REG_SA_CNT_CLR_CE	0x80010
HPI写命令计数	REG_SA_HPI_WR_CNT	0x80E00
HPI读命令计数	REG_SA_HPI_RD_CNT	0x80E04



统计名称	统计寄存器名	偏移地址
SA AXI读PPS统计	REG_SA_AXI_RD_PPS_CNT	0x80E10 0x80E14
SA AXI读BPS统计	REG_SA_AXI_RD_BPS_CNT	0x80E18 0x80E1C
SA AXI写PPS统计	REG_SA_AXI_WR_PPS_CNT	0x80E20 0x80E24
SA AXI写BPS统计	REG_SA_AXI_WR_BPS_CNT	0x80E28 0x80E2C
PPM模块收到POE发出的低优先级TASK统计	REG_PPM_RX_LO_TASK_CNT	0x90E00
PPM模块收到POE发出的高优先级TASK统计	REG_PPM_RX_HI_TASK_CNT	0x90E04
PPM模块收到POE发出的配表TASK统计	REG_PPM_RX_TB_TASK_CNT	0x90E08
PPM模块BD_RD发出BD个数统计	REG_PPM_BD_RD_TX_BD_CNT	0x90E0C
PPM模块发出TASK读次数统计	REG_PPM_TX_TASK_RD_CNT	0x90E10
PPM模块收到无效TASK个数统计	REG_PPM_RX_INVLD_TASK_CNT	0x90E14
PPM模块收到错误TASK个数统计	REG_PPM_RX_ERR_TASK_CNT	0x90E18
PPM模块收到BLK MODE报文统计	REG_PPM_RX_BLK_M_PKT_CNT	0x90E40
PPM模块收到错误BD个数统计	REG_PPM_RX_ERR_BD_CNT	0x90E44
PPM模块向RPM写入INFO个数统计	REG_PPM_WR_RPM_INFO_CNT	0x90E48
PPM模块读取BUF INFO统计	REG_PPM_RD_BUF_INFO_CNT	0x90E50



统计名称	统计寄存器名	偏移地址
PPM模块读取BUF统计	REG_PPM_RD_BUF_CNT	0x90E54
PPM模块收到BUF INFO统计	REG_PPM_RX_BUF_INFO_CNT	0x90E60
PPM模块收到BUF统计	REG_PPM_RX_BUF_CNT	0x90E64
PPM模块向CPM发出预取请求个数统计	REG_PPM_TX_CPM_PREFETCH_CNT	0x90E70
PPM模块向CPM发送BD个数统计	REG_PPM_TX_CPM_BD_CNT	0x90E74
PPM模块读报文统计	REG_PPM_RD_PKT_CNT	0x90E80
PPM模块发送CPM报文个数统计	REG_PPM_TX_CPM_PKT_CNT	0x90E84
PPM模块收到报文统计	REG_PPM_RX_PKT_CNT	0x90E88
PPM模块TBM收到配置个数统计	REG_PPM_TBM_RX_CFG_CNT	0x90EF0
PPM模块TBM收到读配置个数统计	REG_PPM_TBM_RX_RD_CNT	0x90EF4
PPM模块TBM收到带LAST标记配置个数统计	REG_PPM_TBM_RX_LAST_CNT	0x90EF8
PPM模块TBM收到配置字节数统计	REG_PPM_TBM_RX_CFG_BYTE_CNT	0x90EFC
PPM模块TBM发送CQE个数统计	REG_PPM_TBM_TX_CQE_CNT	0x90F00
PPM模块TBM发送回读报文个数统计	REG_PPM_TBM_TX_PKT_CNT	0x90F04
PPM模块TBM发出配置写个数统计	REG_PPM_TBM_CFG_WR_CNT	0x90F10
PPM模块TBM发出配置读个数统计	REG_PPM_TBM_CFG_RD_CNT	0x90F14
PPM模块TBM配置读响应个数统计	REG_PPM_TBM_CFG_ACK_CNT	0x90F18
PPM模块ADD POE处理的POE统计	REG_PPM_PROC_POE_CNT	0x90F30



统计名称	统计寄存器名	偏移地址
PPM模块ADD POE统计	REG_PPM_ADD_POE_CNT	0x90F50
PPM模块POE返回credit数量统计	REG_PPM_POE_RTN_CREDIT_CNT	0x90F54
PPM模块收到VF低优先级报文统计	REG_PPM_RX_VF_LO_PKT_CNT	0x42E20
PPM模块收到VF高优先级报文统计	REG_PPM_RX_VF_HI_PKT_CNT	0x42E24
PPM模块收到VF低优先级报文字节统计	REG_PPM_RX_VF_LO_BYTE_CNT	0x42E30
		0x42E34
PPM模块收到VF高优先级报文字节统计	REG_PPM_RX_VF_HI_BYTE_CNT	0x42E38
		0x42E3C
CPM模块处理新流有效的BD统计	REG_CPM_PROC_NEW_FLAG_CNT	0x98E00
CPM模块处理跨包不使能的BD统计	REG_CPM_PROC_CRS_DISABLE_CNT	0x98E04
CPM模块往BMF模块发送报文统计	REG_CPM_SND_BMF_PKT_CNT	0x98E18
CPM模块往BMF模块发送MBUF统计	REG_CPM_SND_BMF_MBUF_CNT	0x98E1C
CPM模块读流表字节统计	REG_CPM_FLOW_RD_BYTE_CNT	0x98E20
CPM模块读链表字节统计	REG_CPM_LINK_RD_BYTE_CNT	0x98E24
CPM模块写流表字节统计	REG_CPM_FLOW_WR_BYTE_CNT	0x98E28
CPM模块写链表字节统计	REG_CPM_LINK_WR_BYTE_CNT	0x98E2C
CPM 低优先级通道同流阻塞时间统计	REG_CPM_LO_BUSY_TIME_CNT	0x98E30
CPM 高优先级通道同流阻塞时间统计	REG_CPM_HI_BUSY_TIME_CNT	0x98E34
BMF模块接收mbuf eom个数统计	REG_BMF_RX_EOM_CNT	0xA0E04



统计名称	统计寄存器名	偏移地址
BMF模块发送给PMU模块的MBUF个数统计	REG_BMF2PMU_EOM_CNT	0xA0E08
BMF模块发送给PMU模块的VEC个数统计	REG_BMF2PMU_VEC_CNT	0xA0E0C
BMF模块6BYTE KEY非BAD KEY的数量	REG_BMF_NOR_KEY0_CNT	0xA0E10
BMF模块3Byte key访问linklist表统计	REG_BMF_3B_LL_RD_CNT	0xA0E14
BMF模块4Byte key访问linklist表统计	REG_BMF_4B_LL_RD_CNT	0xA0E18
BMF BAD KEY 片内linklist表访问次数统计	REG_BMF_BAD0_LL_RD_CNT	0xA0E1C
BMF模块6Byte key访问linklist表统计	REG_BMF_6B_LL_RD_CNT	0xA0E20
BMF模块读取内存读命令统计暨BMF链表返回个数统计	REG_BMF_RD_DDR_CNT	0xA0E24
BMF模块读取内存读长度统计暨BMF链表返回BYTE数统计	REG_BMF_RD_DDR_RLEN_CNT	0xA0E28
BMF模块读内存返回EOP数量统计	REG_BMF_DDR_REOP_CNT	0xA0E2C
BMF CACHE命中统计	REG_BMF_CACHE_HIT_CNT	0xA0E34
BMF CACHE不命中统计	REG_BMF_CACHE_NHIT_CNT	0xA0E38
BMF 3Byte key 超过hash假命中门限后丢弃的结果统计	REG_BMF_3B_DROP_CNT	0xA0E3C
BMF 4/5Byte key 超过hash假命中门限后丢弃的结果统计	REG_BMF_4B_DROP_CNT	0xA0E40
BMF 6/7Byte key 超过hash假命中门限后丢弃的结果统计	REG_BMF_6B_DROP_CNT	0xA0E44
som info数量统计	REG_BMF_SOM_INFO_CNT	0xA0E48
3B KEY EOM数量统计	REG_BMF_3B_EOM_CNT	0xA0E4C
4/5B KEY EOM数量统计	REG_BMF_4B_EOM_CNT	0xA0E50



统计名称	统计寄存器名	偏移地址
6/7B KEY EOM数量统计	REG_BMF_6B_EOM_CNT	0xA0E54
3B KEY key数量统计	REG_BMF_3B_KEY_CNT	0xA0E58
4/5B KEY key数量统计	REG_BMF_4B_KEY_CNT	0xA0E5C
6/7B KEY key数量统计	REG_BMF_6B_KEY_CNT	0xA0E60
ctrl接收到filter的eom个数统计	REG_PMU_CTRL_RC_FILT_EOM_CNT	0xA8E00
ctrl接收到filter 的vec个数统计	REG_PMU_CTRL_RC_FILT_VEC_CNT	0xA8E04
ctrl接收到片外cache的eom个数统计	REG_PMU_CTRL_RC_OFFCHIP_RC_EOM_CNT	0xA8E08
cache接收到片外vec统计	REG_PMU_CACHE_OFF_RX_VEC_CNT	0xA8E10
dma接收到AXI last统计	REG_PMU_CACHE_DMA_RX_LAST_CNT	0xA8E20
dma发送到AXI读命令统计	REG_PMU_CACHE_DMA_READ_CMD_CNT	0xA8E24
读指令数据量统计	REG_PMU_CACHE_READ_INSTR_CNT	0xA8E28
blk接收到的eop个数统计	REG_PMU_BKL_RC_EOP_CNT	0xA8E40
blk接收到的eom个数统计	REG_PMU_BKL_RC_EOM_CNT	0xA8E44
blk接收到send的释放blk_ram次数统计	REG_PMU_BKL_ENG_RC_RELEASE_CNT	0xA8E4C
eng接收到的eom个数统计	REG_PMU_ENG_RC_EOM_CNT	0xA8E60
eng接收到的vec个数统计	REG_PMU_ENG_RC_VEC_CNT	0xA8E64
send发送给cpm跨包信息次数统计	REG_PMU_SEND_TX_CRS_CNT	0xA8E68
send发送到rpm的结果统计	REG_PMU_SEND_RSLT_CNT	0xA8E6C



统计名称	统计寄存器名	偏移地址
send发送到cpm的跨包大于最大值报文统计	REG_PMU_SEND_CRD_ERR_CNT	0xA8E70
send发送到rpm的mid_fifo满报文统计	REG_PMU_SEND_OVFL_ERR_CNT	0xA8E74
send发送到rpm的超时报文统计	REG_PMU_SEND_TIMEOUT_ERR_CNT	0xA8E78
filter接收到eom个数统计	REG_PMU_FILTER_RC_EOM_CNT	0xB0E00
filter接收到vec个数统计	REG_PMU_FILTER_RC_VEC_CNT	0xB0E04
filter接收到需要预过滤vec个数统计	REG_PMU_FILTER_FILTER_EN_CNT	0xB0E08
filter接收到不需要预过滤vec个数统计	REG_PMU_FILTER_NOFILTER_EN_CNT	0xB0E0C
str接收到eom个数统计	REG_PMU_STR_RC_EOM_CNT	0xB0E20
str接收到vec个数统计	REG_PMU_STR_RC_VEC_CNT	0xB0E24
dfa引擎0输入有效vector统计	REG_PMU_DFA0_VEC_CNT	0xB8E00
dfa引擎0收到EOM统计	REG_PMU_DFA0_EOM_CNT	0xB8E04
dfa引擎0空闲统计	REG_PMU_DFA0_IDLE_CNT	0xB8E08
nfa0从vec_fifo中读出eom统计	REG_PMU_NFA0_EOM_CNT	0xC0E00
nfa0从vec_fifo中读出vec统计	REG_PMU_NFA0_VEC_CNT	0xC0E04
nfa0的ptr统计	REG_PMU_NFA0_PTR_CNT	0xC0E08
nfa0的idle状态拍数统计	REG_PMU_NFA0_IDLE_CNT	0xC0E0C
nfa0副引擎申请指令次数统计	REG_PMU_NFA0_SLA_REQ_CNT	0xC0E10
nfa0的超时统计	REG_PMU_NFA0_VEC_TIMEOUT_CNT	0xC0E14



统计名称	统计寄存器名	偏移地址
nfa0的超时统计	REG_PMU_NFA0_TCP_TIME_OUT_CNT	0xC0E18
nfa0的mid_fifo满统计	REG_PMU_NFA0_MID_OVFL_CNT	0xC0E1C
nfa0命中配置的分组次数	REG_PMU_NFA0_CHK_HIT_CNT	0xC0E20
nfa0命中配置的分组的处理时间统计	REG_PMU_NFA0_CHK_PRO_CNT	0xC0E24
RPM模块接收tcp报文sop个数统计	REG_RPM_RX_TCP_SOP_CNT	0xC8E00
RPM模块接收tcp报文eop个数统计	REG_RPM_RX_TCP_EOP_CNT	0xC8E04
RPM模块匹配规则数超过最大配置值的task个数统计	REG_RPM_MATCH_RLT_THAN_CONF_CNT	0xC8E0C
RPM模块匹配结果超出最大配置值后丢弃的匹配结果个数统计	REG_RPM_SEND_MRESULT_RMAX_DROP_CNT	0xC8E10
RPM模块因去重丢弃的rslt个数统计	REG_RPM_MATCH_DIS_CNT	0xC8E14
RPM模块去重失败个数统计	REG_RPM_MATCH_FAIL_CNT	0xC8E18
RPM模块任务超时个数统计	REG_RPM_TASK_TIMEOUT_CNT	0xC8E1C
RPM模块发送数据到AXI统计	REG_RPM_RSLT_CYCLE_CNT	0xC8E20
RPM模块func发送的有匹配结果的报文个数统计	REG_RPM_FUNC_SEND_MATCHED_PKT_CNT	0x49E00
RPM模块func输出结果个数统计	REG_RPM_FUNC_OUT_RSLT_CNT	0x49E04

各模块入口报文个数统计应相等：

REG_PPM_RX_LO_TASK_CNT (0x90E00)
+ REG_PPM_RX_HI_TASK_CNT (0x90E04)
- REG_PPM_RX_ERR_BD_CNT (0x90E44)
= REG_PPM_TX_CPM_PKT_CNT (0x90E84)
= REG_CPM_SND_BMF_PKT_CNT (0x98E18)
= REG_RPM_RX_TCP_SOP_CNT (0xC8E00)



= REG_RPM_RX_TCP_EOP_CNT (0xC8E04)
= REG_PPM_PROC_POE_CNT (0x90F30)
- REG_PPM_RX_ERR_BD_CNT (0x90E44)
- REG_PPM_RX_TB_TASK_CNT (0x90E08)。

5.9.2.2.2 性能统计

提供SA匹配处理时延统计功能，提供指定子库时延统计、同步时延统计和异步时延统计。

指定子库时延统计：

- 通过寄存器REG_RPM_DELAY_LIB_ID (0xC8008) 配置需要进行时延统计的子库ID。
- 将寄存器REG_RPM_LIB_DELAY_EN (0xC8000) bit[0]先写0后写1，即可将之前的时延统计清零，并开始启动新的时延统计。
- 发包，在发包过程中，通过寄存器REG_RPM_CHK_LIB_ID_MAX_DELAY (0xC8B10)、REG_RPM_CHK_LIB_ID_MIN_DELAY (0xC8B14) 可以读出指定流的最大时延和最小时延值，单位为0.128μs。
- 平均时延计算：停包，通过寄存器REG_RPM_CHK_LIB_ID_ALL_DELAY (0xC8B18) 得到指定流总延时值（单位为0.128μs），通过寄存器REG_RPM_CHK_LIB_ID_TCP_CNT (0xC8B40) 得到指定流时延统计包数，通过指定流总延时值/指定流时延统计包数 得到平均时延值。

同步时延统计：

- 将寄存器REG_RPM_ALL_DELAY_EN (0xC8004) bit[0]先写0后写1，即可将之前的时延统计清零，并开始启动新的时延统计。
- 发同步报文，在发包过程中，通过寄存器REG_RPM_CHK_SYN_MAX_DELAY (0xC8B1C)、REG_RPM_CHK_SYN_MIN_DELAY (0xC8B20) 可以读出全局的最大时延和最小时延值，单位为0.128μs。
- 平均时延计算：停包，通过寄存器REG_RPM_CHK_SYN_ALL_DELAY (0xC8B24) 得到全局总延时值（单位为0.128μs），通过寄存器REG_RPM_CHK_SYN_TCP_CNT (0xC8B44) 得到全局时延统计包数，通过 全局总延时值/全局时延统计包数得到平均时延值。

异步时延统计：

- 将寄存器REG_RPM_ALL_DELAY_EN (0xC8004) bit[0]先写0后写1，即可将之前的时延统计清零，并开始启动新的时延统计。
- 发异步报文，在发包过程中，通过寄存器REG_RPM_CHK_ASYNC_MAX_DELAY (0xC8B28)、REG_RPM_CHK_ASYNC_MIN_DELAY (0xC8B2C) 可以读出全局的最大时延和最小时延值，单位为0.128μs。
- 平均时延计算：停包，通过寄存器REG_RPM_CHK_ASYNC_ALL_DELAY (0xC8B30) 得到全局总延时值（单位为0.128μs），通过寄存器REG_RPM_CHK_ASYNC_TCP_CNT (0xC8B48) 得到全局时延统计包数，通过 全局总延时值/全局时延统计包数得到平均时延值。

5.9.3 典型定位案例

NA。



5.10 注意事项

1. SA使用的部分表项（流表、Linklist表、指令表，跨包信息）存放在内存中，必须物理连续存放。
2. 考虑到内存控制器本身有ECC，存放在内存中的表项不进行校验。
3. 为避免非64B cacheline对齐写入，系统先读64B拼接后再写造成性能损失，SA要求跨包缓存、匹配结果缓存基地址按64B cacheline对齐，并且大小也要为64B整数倍，SA写入时都按64B cacheline对齐写入，以提高性能。
4. 软件分配内存给SA模块使用的Linklist表、指令表、跨包流表的首地址建议以128Byte对齐。



6 SA 规则转换器

6.1 概述

SA转换器采用软件来对正则规则库进行编译，编译为模式匹配引擎的可执行指令，并将模式匹配指令存储在内存中。转换器FS主要供产品设计（软件）、SA算法设计、SA转换器开发人员使用，芯片IP开发和验证人员可不关注转换器章节。

6.2 关键术语

表6-1为与SA相关的关键术语表。

表 6-1 SA 转换器术语表

术语	描述
SA	Service Awareness，业务感知。
AMBA	Advanced Microcontroller Bus Architecture，由ARM公司研发的一种SoC片上系统总线，可将RISC处理器集成在其他IP芯核和外设中。
BD	Buffer Descriptor，描述符，包括发送描述符和接收描述符，描述报文在主存中的存储信息，包括报文起始地址，报文解析结果等等。
Graph	图，对应规则子库。
PPM	packet process manager 报文处理模块，实现报文和报文的预处理。
CPM	cross process manager 跨包处理模块，实现跨包信息的读取和存储，以支持流模式匹配。
BMF	bloom filter 分组过滤模块。
PMU	pattern match unit 模式匹配单元。
RPM	result process manager 结果处理和上报模块。



6.3 特性

SA转换器有如下特性：

- 支持规则库编译。
 - 规则总数不超过10万。
 - 支持PCRE正则语法正确性检查，不正确的语法规则进行报错。
 - 支持对部分输入的规则进行等价改写。
 - 支持多规则编译方式，将含多条正则表达式的规则集编译为一个状态机。
 - 支持将PCRE正则转换为硬件可执行指令。
 - 支持以单条模式串为粒度指定大小写敏感，不支持在单条模式串内分段改变大小写敏感属性。
 - 支持分库编译，分库数量不超过126个，支持多条正则使用同一个rule_id。
- 支持正则规则规模。
 - 单条字符串规则内的显式字符数量不超过1024，超过无法保证编译通过。
 - 支持大括号规则匹配{m, n}，且n<=2047。
 - 支持单条规则中大括号的数量最大为16个。
 - 支持四层以内（包含四层）的大括号嵌套。
- 全部规则进行编译的总时间不大于400s。
- 规则库编译的内存开销小于900MB。
- 转换器DFX。
 - 支持单条规则的语法检查，并提供单独的接口。
 - 提供每个graph（子库）的状态机总大小（指令表所占空间）。
 - 在log提供每个分组内的正则规则数和指令实际占用大小。
 - 支持上报一组可疑的编译失败的规则并给出可能原因，不一定能够准确上报具体哪一条规则编译失败。
- 转换器其他可能编译失败报错。
 - 单分组状态机超大（>2KB）。
 - 单子库过滤模块哈希冲突的关键字数量超规格。
 - 多子库合并时过滤模块合并超规格失败。

6.4 遵循协议

PCRE正则语法，下[表6-2](#)为所支持的正则语法说明。

表 6-2 SA 正则语法支持表

语法	描述
\	将下一个字符标记为一个特殊字符或一个原义字符。例如，“n”匹配字符“n”。“\n”匹配一个换行符。序列“\\”匹配“\”而“\(”则匹配“（”。



语法	描述
^	匹配整个输入字符串的开始位置，当前将会转变为\n的换行符，仅支持出现在正则表达式开头。
\$	匹配整个输入字符串的结束位置，当前将会转变为\n的换行符，仅支持出现在正则表达式结尾。
*	匹配前面的子表达式零次或多次。例如，zo*能匹配“z”、“zo”以及“zoo”。*等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。
{n}	n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。
{n,}	n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooooo”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。当前受限于大括号上限，将其等价于{n,2047}。
{n,m}	m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，“o{1,3}”将匹配“foooooo”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。
.	匹配任何单个字符（包括\n）。
(pattern)	匹配pattern这一子字符串。
x y	匹配x或y。例如，“z food”能匹配“z”或“food”。“(z f)ood”则匹配“zood”或“food”。 1. 不支持 两边出现^或\$的符号，\x26uid\x3d[a-f0-9]{16}(\$ \x26)。 2. 不支持 的一边出现空白的情况，\x2e(cellpadding cellspacing width){0,128}\x3Ctable[^\x3E]id\x3D[\x22\x27]。
[xyz]	字符集合（character class）。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。特殊字符仅有反斜线\保持特殊含义，用于转义字符。其他特殊字符如星号、加号、各种括号等均作为普通字符。脱字符^如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。连字符-如果出现在字符串中间表示字符范围描述；如果出现在首位则仅作为普通字符。
[^xyz]	排除型（negate）字符集合。匹配未列出的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。
[^a-z]	排除型的字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。



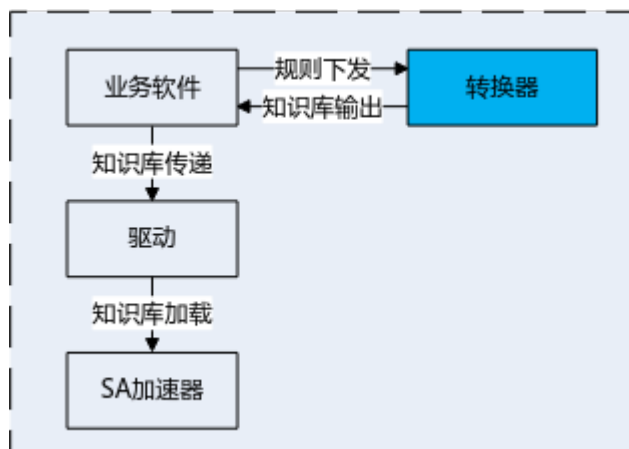
语法	描述
\d	匹配一个数字字符。等价于[0-9]。
\D	匹配一个非数字字符。等价于[^0-9]。
\n	匹配一个换行符。等价于\x0a。
\r	匹配一个回车符。等价于\x0d。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于[^ \f\n\r\t\v]。
\w	匹配包括下划线的任何单词字符。等价于 “[A-Za-z0-9_]” 。
\W	匹配任何非单词字符。等价于 “[^A-Za-z0-9_]” 。
\xn	匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码。
\b	匹配\w字符和\W字符中间位置。例如：[a-z]\b可以匹配a%cd的a和d但是不能匹配c。
\B	非\b。
\a	报警字符（打印它的效果是电脑嘀一声），等价于\x07。
\t	制表符，tab，等价于\x09。
\e	Escape，等价于\x1b。
\v	竖向制表符，等价于\x0b。
\f	换页符，等价于\x0c。

6.5 功能描述



6.5.1 SA 加速器应用图

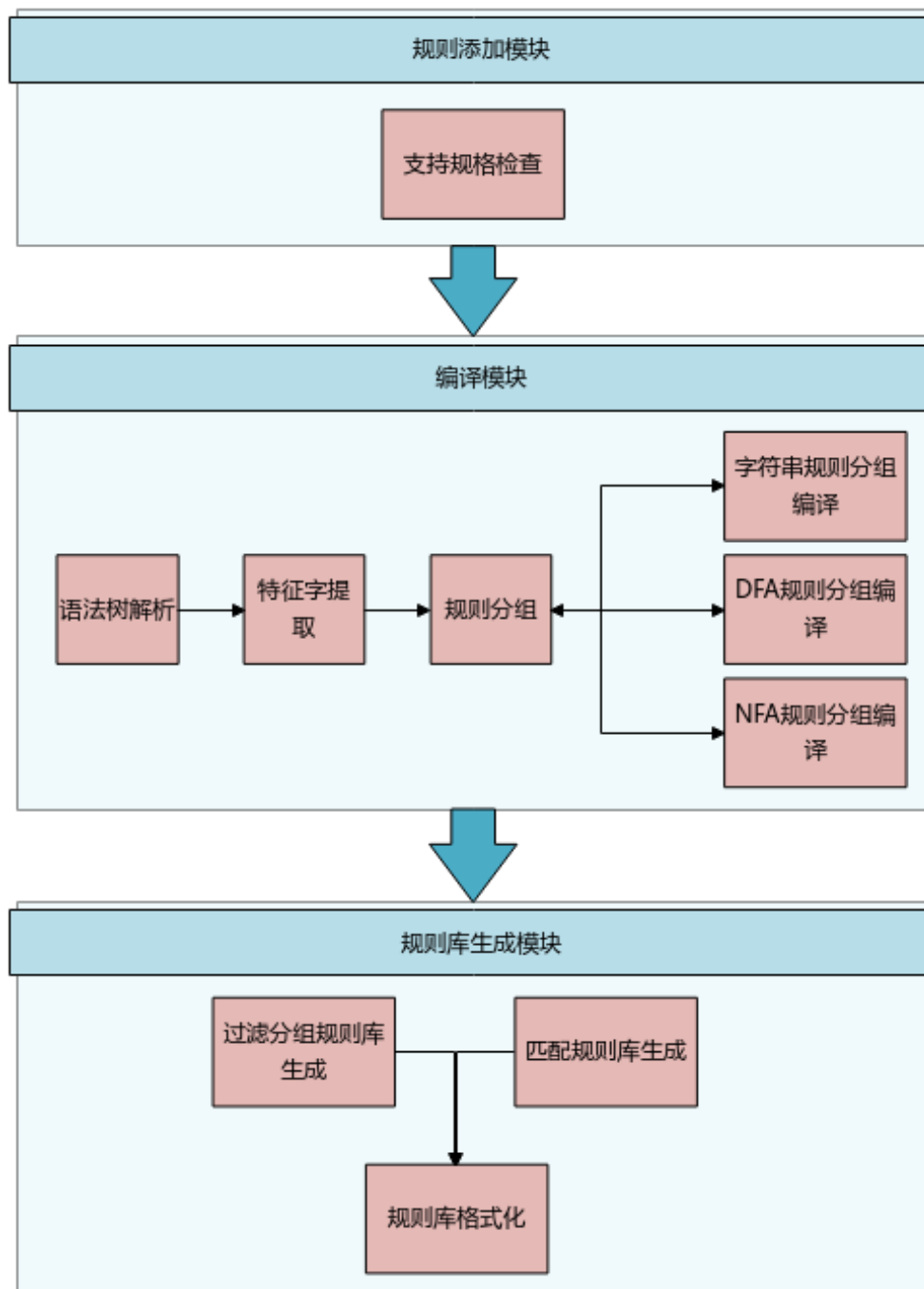
图 6-1 SA 转换器使用示意图



通转换器以函数库的形式提供，供业务软件进行调用。业务软件将规则输入硬件转换器，转换器完成规则到知识库的转换并返回结果，然后由驱动把知识库加载到 SA 加速器，交互过程如上图 6-1 所示。

6.5.2 SA 转换器处理过程

图 6-2 SA 转换器处理流程图



规则添加模块：

- 通过调用规则添加接口函数，添加一个规则子库。
- 逐条规则读取并转换为内部规则格式，判断规格限制，若子库规格不符合要求，则报错。



- 生成一个子库ID并返回给产品，将该规则子库加入待编译规则库中，规则添加完成，等待添加下一组规则子库。

编译模块：

- 规则全部添加完毕后，调用编译函数接口，启动规则编译，逐个子库进行编译过程。
- 进行语法解析，若语法错误则报错返回，否则转换成AST抽象语法树并进行分割优化。
- 对每条规则进行特征字提取，直到全部规则的特征字都提取完毕。
- 根据特征字进行智能分组。
- 对于各分组调用对应类型的编译引擎进行规则编译，对于结果超规格的分组发回分组模块重新进行优化分组。
- 待整个子库编译完成后进行下一个规则子库的编译。
- 全部规则库独立编译完毕后，进行分组过滤规则的整体合并，得到最终的分组过滤表。

规则库生成模块：

- 将编译完成的分组过滤表进行硬件格式的转换。
- 将编译完成的各个匹配分组规则进行硬件格式转换。
- 输出最终的规则库。

6.6 编程接口

6.6.1 规则信息结构体

```
// 存放单个lib规则的信息
typedef struct _RuleInfo {
    unsigned int lib_id;        // valid value: 1~126
    unsigned int rule_cnt;      // number of rules in this lib
    char **pRuleContent;       // rule contents, an array of (char *)
    unsigned int *pID;         // id of each rule, an array of (unsigned int)
    unsigned int *pFlag;        // flag of each rule;
    // (flag & 0x01) == 0 ? 大小写不敏感 : 大小写敏感
    // (flag & 0x02) == 0 ? 去重时关注上报的结束位置 : 去重时不关注上报的结束位置
    // (flag & 0x04) == 0 ? 去重时关注上报的起始位置 : 去重时不关注上报的起始位置
}t_RuleInfo;
```

📖 说明

规则信息说明：

1. 规则主要分为两种类型，字符串和正则表达式。字符串规则中的特殊字符（为区别正则表达式中有特殊含义的特殊字符：*、?、^、\$等）全部由其ASCII码代替，例如：字符“*”在字符串规则中被表示为0x2a。
2. 规则信息中表征规则内容的字符指针以“\0”为结束。
3. 所有规则中的字符都由ASCII码表示。

6.6.2 编译结果结构体

```
// 存放单个表项的内容
typedef struct _Data {
```



```
unsigned int magic;           // 0xdeadbeef
unsigned int crc32;           // 表项数据的crc值
unsigned int lib_id;          // (如果是子库指令表)对应的lib id
unsigned int data_type;       // 表项的类型
unsigned int data_length;     // content的长度
unsigned char pData[0];       // 表项内容
}t_Data;
// 存放最终输出结果的内容
typedef struct _ResultInfo {
char compiler_version[256];   // 转换器的版本
unsigned int magic_word;      // 0xdeadbeef
unsigned int crc32;           // 结果的crc值
unsigned int data_cnt;        // 总的t_Data数量(公共表项+片外指令表)
unsigned int graph_cnt;       // 结果中片外指令表的数量
unsigned int *pGraphID;       // 片外指令表的id, 数组, 长度为graph_cnt
unsigned int *pGraphSize;     // 各子库的大小, 数组, 长度为graph_cnt
t_Data **Datas;              // 所有t_Data指针的数组, 长度为data_cnt
}
```

6.6.3 转换器接口函数定义

6.6.3.1 CheckGrammer

函数原型:

```
unsigned int CheckGrammer(const char *content, unsigned int length);
```

函数功能:

检查单条规则是否符合语法。

输入:

Content: 输入规则

Length: 规则的长度

输出:

unsigned int类型的返回码, 0表示执行成功, 其他值表示执行失败

6.6.3.2 OpenConverter

函数原型:

```
unsigned int OpenConverter(void **pConverter);
```

函数功能:

创建转换器对象, 如果创建成功则返回转换器对象的指针。

输入:

pConverter: 上层传入的转换器空指针的地址

输出:

unsigned int类型的返回码, 0表示执行成功, 其他值表示执行失败

若执行成功则*pConverter为一个合法的转换器对象的指针

6.6.3.3 CloseConverter

函数原型:

```
unsigned int CloseConverter(void **pConverter);
```

函数功能:



关闭转换器，释放转换器所占的空间。

输入：

pConverter：上层传入的合法的转换器指针的地址

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.4 AddRule

函数原型：

```
unsigned int AddRule(void *pConverter, const t_RuleInfo *pRuleInfo);
```

函数功能：

向合法的转换器添加单个lib的规则。

输入：

pConverter：合法的转换器指针

pRuleInfo：单个lib规则的信息

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.5 Convert

函数原型：

```
unsigned int Convert(void *pConverter);
```

函数功能：

执行规则转换的动作。

输入：

pConverter：合法的转换器指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.6 Combine

函数原型：

```
unsigned int Combine(void **pConverterIn, unsigned int number, void *pConverterOut);
```

函数功能：

将多次Convert的中间结果合并成一个中间结果， pConverterIn是输入（由Open-Add-Converter而来，或者是其他Combine的输出）， pConverterOut是输出，需要先经过OpenConverter得到。

输入：

pConverterIn：合法的转换器指针数组

number：待合并的中间结果的个数

pConverterOut：待输出的中间结果指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.7 GetResult

函数原型：

```
unsigned int GetResult(void *pConverter, t_ResultInfo **pResult);
```



函数功能：

在执行完规则转换的动作之后，调用该接口获取转换后的结果。

输入：

pConverter：合法的转换器指针

pResult：待输出的结果指针的指针，其空间由转换器内部申请

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.8 FreeResult

函数原型：

```
unsigned int FreeResult(t_ResultInfo **pResult);
```

函数功能：

释放转换器所产生的结果。

输入：

pResult：待释放的合法结果指针的指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.9 RegisterAllocFree

函数原型：

```
unsigned int RegisterAllocFree(ifep_malloc mem_alloc, ifep_free mem_free)
```

函数功能：（只执行一次既可，放在初始化过程中）。

给转换器注册内存申请和内存释放函数，如果不执行，则默认为glibc的malloc和free。

输入：

ptr：内存申请函数的指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.10 RegisterLogFunc

函数原型：

```
unsigned int RegisterLogFunc(LogFunc log_function);
```

函数功能：

给转换器注册log打印函数，如果ptr为NULL或者不执行，则默认log不打印。

输入：

ptr：打印函数指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.11 RegisterDbgFunc

函数原型：

```
unsigned int RegisterDbgFunc(DbgFunc dbg_function);
```




函数功能：

给转换器注册debug打印函数，如果ptr为NULL或者不执行，则默认debug不打印。

输入：

ptr：打印函数指针

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.12 RegisterLinkListDepth

函数原型（只需要执行一次即可，可放在初始化过程中）：

```
unsigned int RegisterLinkListDepth(unsigned int Depth);
```

函数功能：

调整生成的linklist表项深度，不调用时默认linklist表项深度为128K。0:128k，1:64k，other:128k。

输入：

Depth：生成的结果中linklist表项的深度

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.13 RegisterKeeyModel

函数原型（只需要执行一次即可，可放在初始化过程中）：

```
unsigned int RegisterKeeyModel(unsigned int model);
```

函数功能：

调整转换器内部选key的长度，默认按照3/5/7的长度选key。

输入：

model：选key的几种模式，0:3/5/7; 1:3/5/6; 2:3/4/7; 3: 3/4/6

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.14 RegisterBadkeyPath

函数原型（只需要执行一次即可，可放在初始化过程中）：

```
unsigned int RegisterBadkeyPath(const char *path, unsigned int length);
```

函数功能：

设置badkey文件的路径（路径中包含文件名），不调用时默认为”./bad_key.key”。

输入：

path：badkey文件的路径（含文件名）

length：路径中字符的长度

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.15 SetDfaThreshold

函数原型：

```
unsigned int SetDfaThreshold(void *pConverter, unsigned int threshold);
```



函数功能：

设置DFA状态机和STR状态机之间的字符长度阈值，不调用时默认阈值为10。

输入：

pConverter：需要改变DFA/STR规则的字符长度阈值的转换器指针

threshold：区分DFA和STR状态机的字符长度阈值

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.16 SetLinkListWidth

函数原型：

```
unsigned int SetLinkListWidth(void *pConverter, unsigned int width);
```

函数功能：

调整生成的linklist表项宽度，不调用时默认linklist表项宽度为16。0:16，1:32，2:64，other:16。

输入：

pConverter：转换器指针

width：生成的结果中linklist表项的宽度

输出：

unsigned int类型的返回码，0表示执行成功，其他值表示执行失败

6.6.3.17 SetResourceLimit

函数原型（每次调用Converter时分别指定）：

```
unsigned int SetResourceLimit(void *pConverter, unsigned int limit);
```

函数功能：

该接口对于309a无任何作用，仅用于与1280保持接口一致。

6.6.3.18 RegisterMemChkFunc

函数原型：

```
unsigned int RegisterMemChkFunc(MemCheck chk_function);
```

函数功能：

注册内存检查函数。内存检查函数由外部提供，该函数的入参已字节为单位，内部检查该函数的返回值为0时则往下继续执行，否则返回内存不足的错误码。

输入：

chk_function：内存校验的函数指针

6.6.3.19 GetDataCrc

函数原型：

```
unsigned int GetDataCrc(const t_Data *tdata);
```

函数功能

计算t_Data的CRC值。



输入：
tdata: 需要计算的tdata指针

6.6.3.20 GetResultCrc

函数原型：
unsigned int GetResultCrc(const t_ResultInfo *result_out);

函数功能
计算t_ResultInfo的CRC值

输入：
result_out: 需要计算的t_ResultInfo指针

6.6.3.21 SetOptimizeToggle

函数原型：
unsigned int SetOptimizeToggle(unsigned int toggle);

函数功能：
内部优化选项的开关，bitmap表示，默认全开。

输入：
toggle: bitmap表示的内部优化的开关选项，当前已有的优化开关如下：

#define OPT_NONFIX_TO_DFA_EN	(0x01)
#define OPT_SPLIT_PFA_OR_EN	(0x02)
#define OPT_KEY_PRIORITY_LFIX_EN	(0x04)
#define OPT_KEY_PRIORITY_KHASH_EN	(0x08)
#define OPT_POP_PFA_EN	(0x10)
#define OPT_ENABLE_PFA	(0x20)
#define OPT_MERGE_CONCAT_EN	(0x40)

6.7 应用配置

图 6-3 规则转换调用流程框图

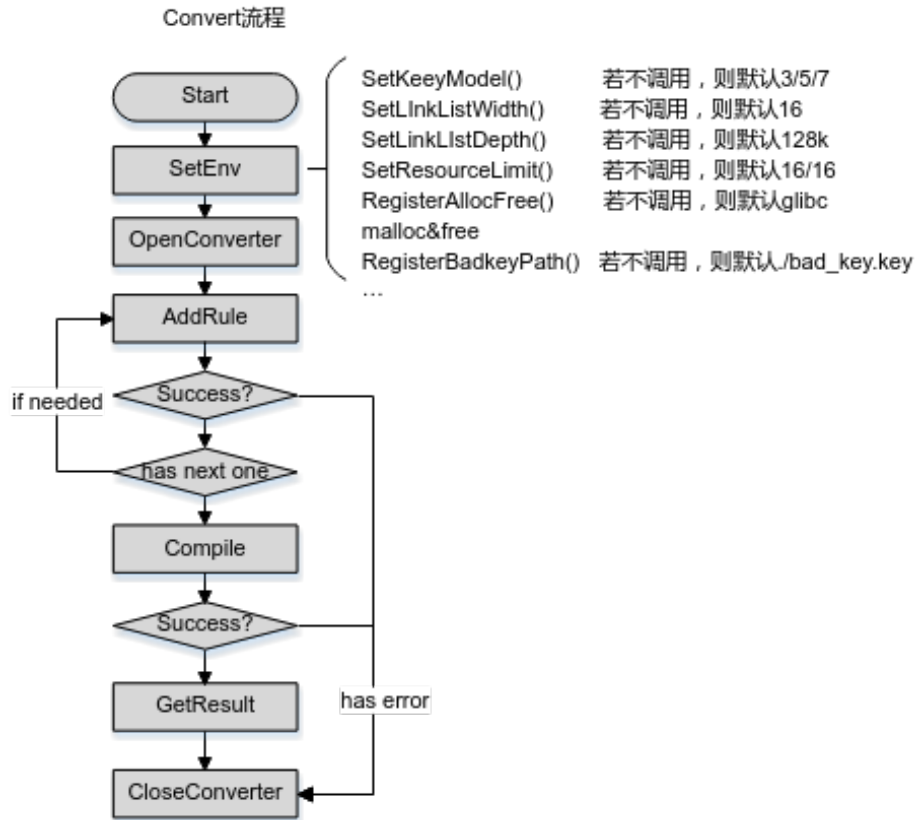
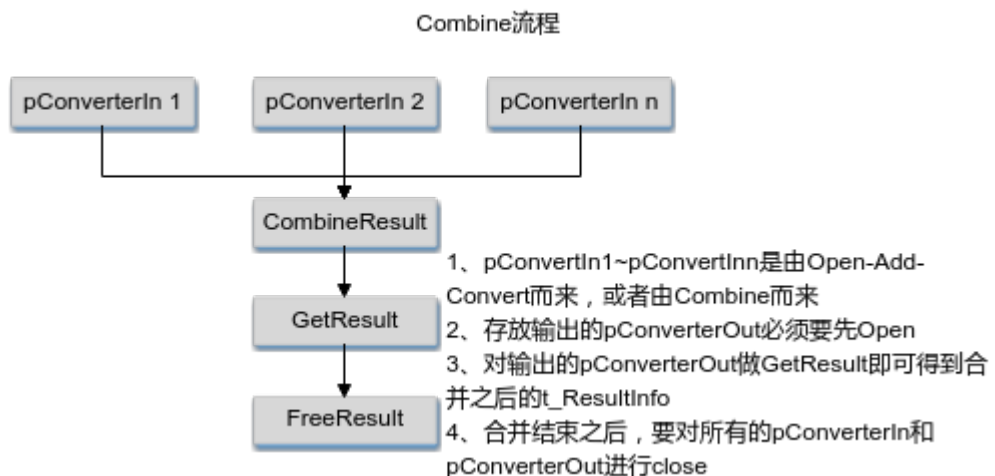


图 6-4 中间结果合并调用流程框图





6.8 返回码说明

表 6-3 返回码说明

错误码值	错误码类型
0	RET_SUCCESS
1	RET_FAILUR
2	RET_BAD_ALLOC
3	RET_INPUT_NULL_PTR
4	DYNAMIC_CAST_ERR
5	CANNOT_VISIT_TYPE
6	UNVALID_PARAM
10	INVALID_MEMORY_ACCESS
11	RULE_HAVE_NO_KEY
12	RULE_OVERSIZE
13	LINKLIST_HASH_FULL
14	RE_EXHAUST_LL_BC
15	RE_EXHAUST_LL_BK
16	UNVALID_INPUT_IN_COMBINE
17	COMBINE_FAILED
18	GROUP_FAILED
19	BK_GROUP_FAILED
20	MEM_CHECK_ERROR
21	RESOURCE_LIMIT_NOT_EQ
1001	RULEINFO_ILLEGAL_RULECNT
1002	RULEINFO_ILLEGAL_LIBID
1003	RULEINFO_ILLEGAL_CONTENT
1004	RULEINFO_ILLEGAL_TOOLARGE
1005	CHECK_A_NULL_STRING
1006	CHECK_A_TOO_LARGE_STRING
2001	CM_LIB_ID_EXISTED



错误码值	错误码类型
3001	RULE_SYNTAX_ERROR
3002	UNVALID_BRACE_VALUE
3003	EMPTY_OBJ_NODE
3004	TOO_MANY_BRACES
3005	TOO_DEEP_BRACES
3006	TOO_LARGE_BRACE_NUM
3007	SHOULD_NOT_APPEAR
5001	STR_GROUP_OVERSIZE
5002	NFA_GROUP_OVERSIZE
5003	DFA_GROUP_OVERSIZE



7 TRING

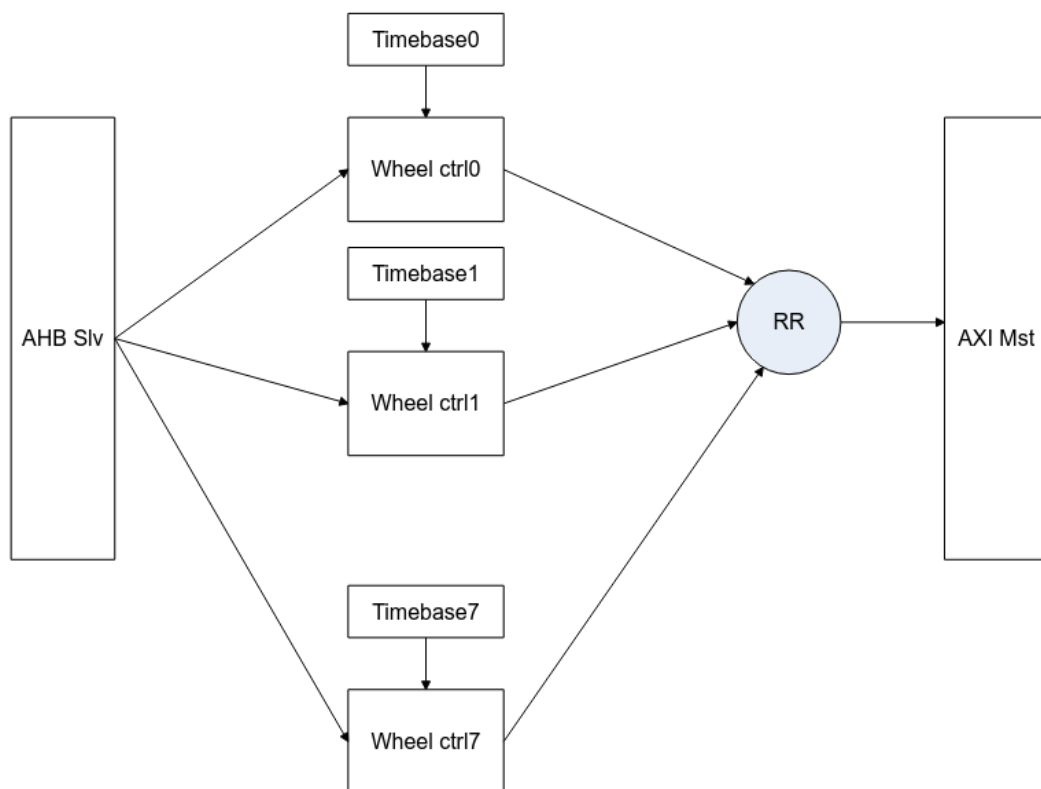
7.1 概述

TRING模块主要是帮助CPU处理定时任务，并且在处理完成之后通知CPU。

TRING提供8个时间轮WHEEL，每个时间轮支持精度独立可配置。

TRING的逻辑实现框图如图7-1所示，每个WHEEL依靠一个独立的时间基准来进行定时时间的计算，每个WHEEL都可以接受CPU下发到TRING slave接口的命令，每个WHEEL对外部内存的操作或者发送POE的消息操作是通过一个RR调度完成。

图 7-1 TRING 结构框图





7.2 关键术语

表7-1为与TRING相关的关键术语表。

表 7-1 TRING 术语表

术语	描述
TRING	Timer Ring，硬件定时环。
TCB	Timer Control Block，定时器控制块。
WHEEL	Wheel，时间轮。
Step_Hdr	Step Header，定时器挂接点。

7.3 特性

TRING模块具有以下特点：

- 支持8个WHEEL独立工作无耦合。
- 每个WHEEL的定时器挂接点Step_Hdr个数可配置，配置范围从1~65535。
- 每个WHEEL上相邻Step_Hdr之间的时间间隔可配置，最小间隔为10μs，最大为255*10μs。
- 每个Step_Hdr上可以挂接多个TCB，各个TCB以双向链表形式连接在一起。
- 支持软件下发（Start）启动新的定时器任务，TRING根据下发的消息读出TCB，把TCB挂接到指定WHEEL的对应的Step_Hdr链表的尾部，启动的定时器必须是已经创建好的定时器。
- 支持软件下发（Resume）重启定时器任务，TRING根据下发的消息中指定的TCB，把TCB中的暂停指示标记清0，Resume的定时器必须是Pause的定时器。
- 支持软件下发（Pause）暂停定时器任务，TRING根据下发的消息把指定的TCB中暂停指示标记为1，TCB依然继续在时间轮上挂接，但是不发送POE消息，也不发送中断。
- TRING中的一个定时器任务到期处理时，根据TCB的具体信息决定是否发送消息给POE或者上报完成中断，添加POE的Group和队列由TCB中信息决定，添加POE消息格式和全局数据结构中描述一致。
- 在处理某个Step_Hdr链条超时，继续处理本链条直到处理结束，处理结束之后，等到下一个时间间隔到来时再去处理下一个Step_Hdr链条。
- 支持多模特性1PF+8VF，支持WHEEL按照模式划分，不同模式的定时器挂接到不同模式的WHEEL上。
- 支持各模式内的TCB资源共享，TCB序列号统一，TCB的虚拟地址连续。
- TRING添加POE消息，固定ADD_POE0；支持按模式假满重传，真满丢弃，且上报可屏蔽的中断，模式之间互不影响。
- TRING读取DDR中内容支持大端可配置。
- 支持各模式内的TCB资源共享，TCB序列号统一，TCB的虚拟地址连续。



- 支持AXI Master OOO接口，支持Outstanding可配置。
- 支持AHB Slave接口，当TRING定时器工作和软件下发任务冲突时，TRING确保操作的原子性，即完成正在完成的任务，再处理另一个任务。
- 不支持Create（创建）定时器，创建定时器由软件完成，当创建一个定时器时，软件把空闲TCB的头指针指到该创建定时器的下一个空闲TCB。
- 不支持Delete（删除）定时器，删除定时器由软件完成，当删除一个定时器时，软件把空闲TCB的头指针指到该删除定时器，并且把该删除定时器的下一个指向原始的空闲头TCB。
- 不支持（Change）修改定时器操作，建议软件采用Stop+软件修改TCB+Start组合来替代。

7.4 遵循协议

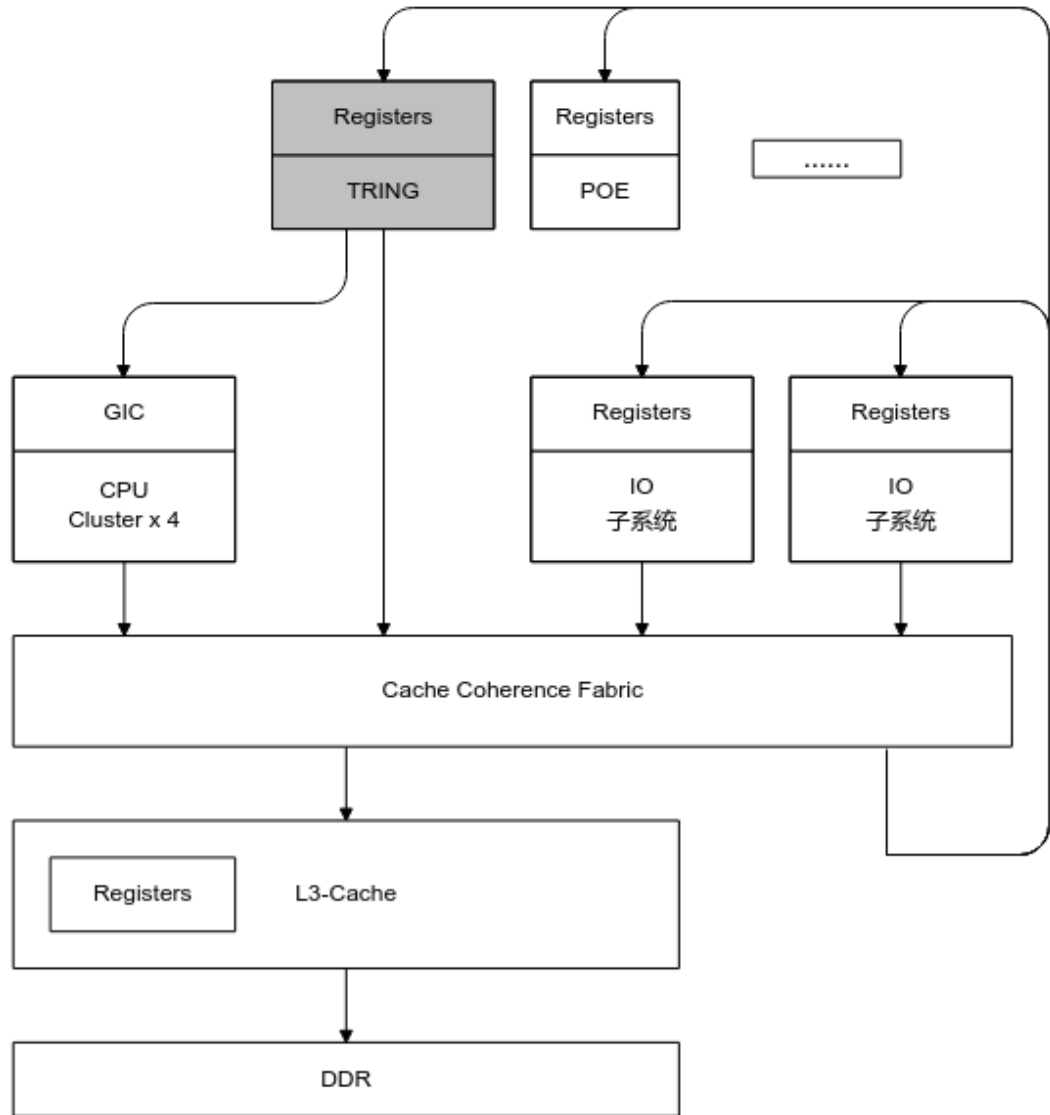
遵循AMBA4.0协议。

7.5 功能描述

软件首先在DDR开辟一段连续的虚拟地址空间（在真实的物理内存地址空间上是连续或者不连续的空间），并切分成一系列大小相同的块，每一个块是一个TCB（timer control block），之后软件会通知TRING模块定时处理这些TCB，TRING中定时时间到之后，处理对应定时周期的TCB，从内存中读取TCB，对TCB部分域（与TRING处理相关的域）做有效性检查，并且根据TCB的指示给POE发消息或者上报完成中断。

7.5.1 典型应用

图 7-2 TRING 应用框图



在系统中，TRING模块会和下面模块进行交互，分别描述如下：

- CPU：通过CPU向TRING发送Start或者Stop命令。
- POE：CPU下发任务时，先发送到POE，然后POE发一个通知消息给TRING，TRING就主动从POE获取任务；TRING在处理完TCB之后，给POE发送消息。
- GIC：TRING在处理TCB任务时发生异常或者完成TCB任务后需要上报完成中断，给GIC或者GIC系统（包括MBI_GEN/ITS）发中断。

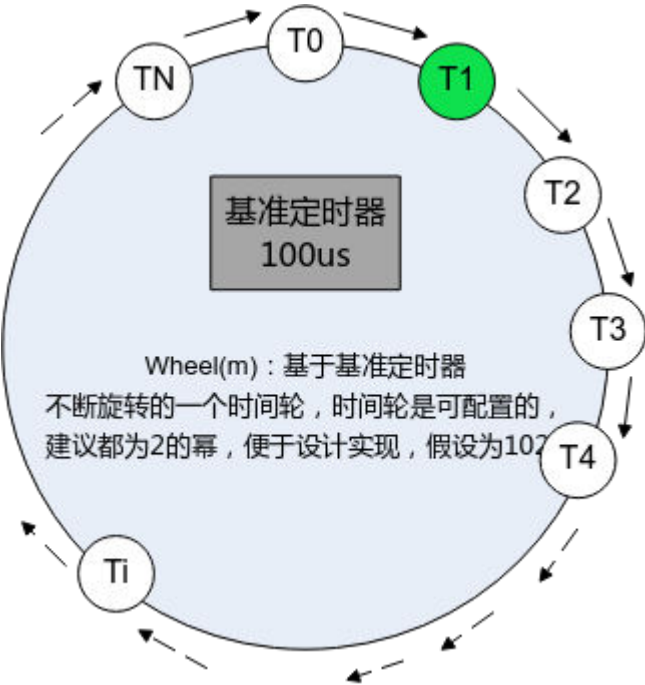
7.5.2 TRING 功能原理

TRING的功能是对内存中的任务进行定时处理。具体说来就是TRING的每个时间轮 WHEEL将时间按一定间隔进行划分，划分成多个Step_Hdr，CPU在某个时间点的队列



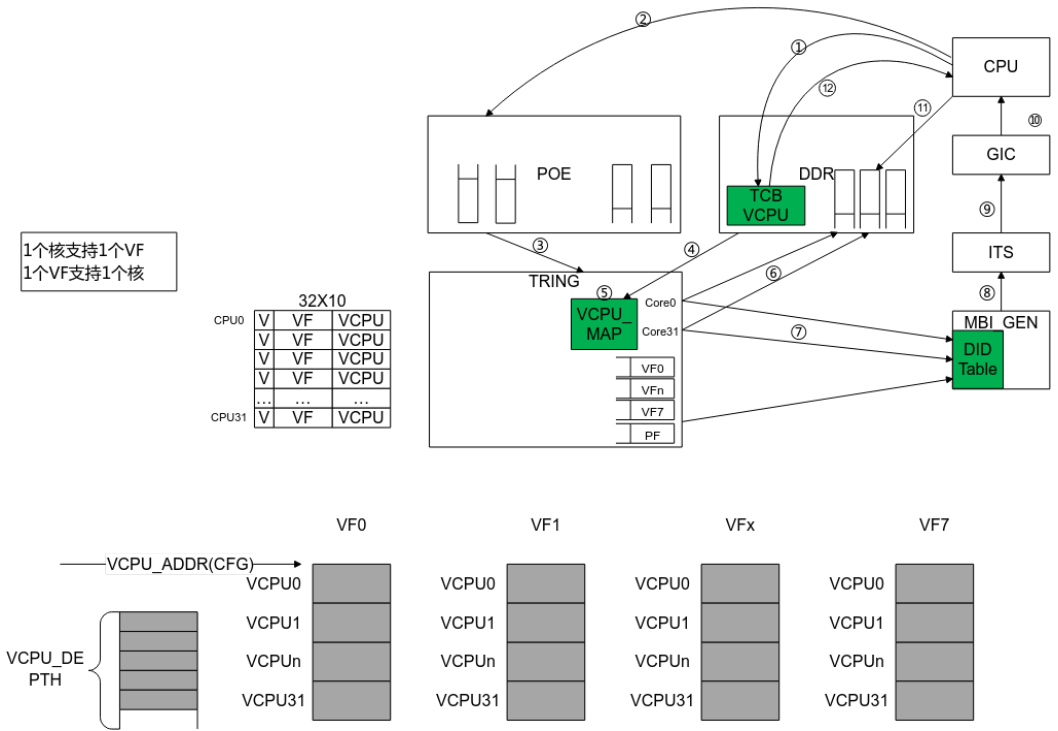
中加入任务。定时器对当前时间进行计数，当计数达到一个新的时间点，将该时间点的任务全部提交给任务管理单元。CPU还可以取消某个任务。

图 7-3 TRING WHEEL 实现原理



TRING在完成TCB任务后，会选择上报POE消息或者完成中断。上报完成中断的步骤如下。

图 7-4 TRING 上报完成中断图





7.5.3 多模隔离

每个TRING WHEEL的寄存器空间最大为64kB，实际使用小于4kB。通过页表划分，将多个模式对TRING WHEEL访问映射到不同的基地址，从而实现TRING WHEEL在物理上的隔离。

须知

一个TRING WHEEL仅能分配给一种模式使用，否则会导致计数值错乱影响软件运行。

7.5.4 原子特性

无。

7.6 软件指南

7.6.1 PF 初始化

TRING在开始使用时，首先根据PF做一些资源分配，以及一些全局的配置。

步骤1 确保TRING复位解除，通过配置相关的寄存器实现，具体请看[软复位](#)中描述。

步骤2 确保TRING的时钟门控打开，通过配置相关的寄存器实现，具体请看[时钟门控](#)中描述。

须知

每次解复位和开启时钟之后，必须读取CRG状态寄存器确认模块已经解复位和开时钟，然后启动模块功能。

TRING时钟状态寄存器，循环查询POE_subctrl的SC_TRING_CLK_ST寄存器（偏移地址0x5378），各有效域段为全1，代表时钟打开。

TRING复位状态寄存器，循环查询POE_subctrl的SC_TRING_RESET_ST寄存器（偏移地址0x5A78），各有效域段为全0，代表复位解除。

连续10μs读取SC_TRING_CLK_ST寄存器非全1，SC_TRING_RESET_ST寄存器非全0，视为异常。

步骤3 给每个VF分配WHEEL资源，配置虚拟WHEEL和物理WHEEL的对应关系。

步骤4 配置各个VF的offset。

步骤5 配置中断使能。

步骤6 配置总线相关的寄存器：包括streamid的模式，PF的streamid值，各个VF的streamid值，tring总线控制寄存器。

步骤7 配置添加POE的相关寄存器：包括配置PF添加POE的地址，各个VF添加POE的地址，以及添加POE的node_id和假满重传次数等。



步骤8 配置相关的页表和中断路由信息。

----结束

7.6.2 软复位

在TRING指示可以复位的情况下，可以对TRING工作逻辑进行软复位，不会对系统造成影响。

步骤1 配置CMM_TO_BE_RST为1，等待内部复位准备完成。

步骤2 查询CMM_COULD_BE_RST寄存器为1时，可以安全复位。

步骤3 通过配置系统控制器中SC_TRING_RESET_REQ为1，可实现对TRING工作逻辑的单独软复位。

步骤4 通过配置系统控制器中SC_TRING_RESET_DREQ为1，可实现对TRING工作逻辑的单独软复位撤离。

----结束

7.6.3 VF 卸载

在TRING工作时对单VF卸载，遵循以下步骤。

步骤1 软件停止此VF任务的下发。

步骤2 配置此VF对应的wheel_enable为0。

步骤3 查询wheel_pend寄存器直到此VF的所有wheel均处于空闲状态。

步骤4 软件回收此VF上的所有TCB并且刷新Step-Hdr的valid域为0。

----结束

须知

如果卸载后属于该VF的WHEEL需要重新分配给其他VF使用，需执行如下操作：

1. PF配置Tring寄存器WHEEL_MAP，给每个VF分配WHEEL资源，重新设置虚拟WHEEL和该物理WHEEL的对应关系。
2. VF按照“VF初始化描述”，完成该WHEEL的初始化。
3. 所有的单模复位都不会复位相应制式的寄存器配置值、状态与统计值，如果该VF被重新分配，需要软件在重新配置之前将这些寄存器清除或配置正确。

7.6.4 时钟门控

上电时TRING工作时钟默认为打开状态。在不使用TRING的情况下，可以关闭TRING工作时钟，以降低芯片功耗。

打开门控时钟具体配置步骤如下：

步骤1 读系统控制器寄存器SC_TRING_CLK_ST，获取当前时钟门控信息。

步骤2 如果SC_TRING_CLK_ST为低，进入**步骤3**；SC_TRING_CLK_ST为高，表示TRING时钟已经打开，直接跳过下面的步骤结束操作。



步骤3 配置系统控制寄存器SC_TRING_CLK_EN为高，打开TRING模块的时钟门控。

----结束

7.6.5 关闭时钟

关闭门控时钟具体配置步骤如下：

步骤1 读系统控制器寄存器SC_TRING_CLK_ST，获取当前时钟门控信息。

步骤2 如果SC_TRING_CLK_ST为高，进入**步骤3**；SC_TRING_CLK_ST为低，表示TRING时钟已经关闭，直接跳过下面的步骤结束操作。

步骤3 配置系统控制寄存器SC_TRING_CLK_DIS为高，打开TRING模块的时钟门控。

----结束

7.6.6 VF 初始化

在加载时，需要对定时环进行初始化：

步骤1 软件开辟好一段连续的TCB虚拟空间，把FreeListPtr的指针指向第一个空闲TCB的虚拟地址，把各TCB中的头尾指针首位相连成一个链，第一个TCB的Pre_idx和最后一个TCB的Nxt_idx都设置为NULL，软件把TCB空间的虚拟起始地址配置到TRING的相关寄存器中。

步骤2 软件开辟好各个WHEEL的Step_Hdr虚拟空间，把所有Step_Hdr的起始头尾指针都初始化为NULL，然后把各WHEEL的Step_Hdr空间的虚拟起始地址配置到TRING的相关寄存器中。

步骤3 初始化定时环配置，包括配置各个WHEEL的时间间隔参数，Step_Hdr的个数（要小于等于内存中相应的Step_Hdr的个数）。

步骤4 配置TRING使能WHEEL寄存器启动所需要的定时器环。

----结束

7.6.7 Create 一个定时器

全部由软件完成，步骤如下：

步骤1 软件从TCB FreeListPtr的头指针获取到空闲头TCB的地址，读取TCB的信息。

步骤2 软件把相关的定时器信息填入这个空闲TCB，把State域由0b00（Free）更新为0b01（Created），TCB的头尾指针不需要关心。

步骤3 软件把TCB FreeListPtr的头指针刷新成当前TCB的Nxt-idx。

----结束

7.6.8 Start 一个定时器

由软件通知TRING完成，Start的定时器必须是已经Create好的：

步骤1 软件把需要启动的定时器TCB的地址，放在发给TRING的消息中发给TRING（实际上是发给POE指定的Group中）。

步骤2 TRING从POE获取到任务消息。



- 步骤3 TRING解析消息，获取到TCB的序号。
- 步骤4 TRING根据Start命令中的TCB(x)序号以及TCB的起始地址，去读取TCB(x)。
- 步骤5 TRING根据TCB内容获取到Wheel_No。
- 步骤6 TRING挂接定时器，写State位为0b11(Running)。

----结束

7.6.9 Resume 一个定时器

由软件通知TRING完成，Resume的定时器必须是已经Pause的：

- 步骤1 软件把需要Resume的定时器TCB的地址，放在发给TRING的消息中发给TRING（实际上是发给POE指定的Group中）。
- 步骤2 TRING从POE获取到任务消息。
- 步骤3 TRING解析消息，获取到TCB的序号。
- 步骤4 TRING根据Resume命令中的TCB(x)序号以及TCB的起始地址，把TCB(x)中的暂停位清0，接着按照Stop处理流程完成此TCB的STOP处理。
- 步骤5 TRING完成Resume_STOP处理后，紧接着按照Start处理流程完成此TCB的START处理。

----结束

7.6.10 Pause 一个定时器

由软件通知TRING完成，Pause的定时器必须是已经Start的：

- 步骤1 软件把需要启动的定时器TCB的地址，放在发给TRING的消息中发给TRING（实际上是发给POE指定的Group中）。
- 步骤2 TRING从POE获取到任务消息。
- 步骤3 TRING解析消息，获取到TCB的序号。
- 步骤4 TRING根据Pause命令中的TCB(x)序号以及TCB的起始地址，把TCB(x)中的暂停位置1。

----结束

7.6.11 Stop 一个定时器

由软件通知TRING完成，Stop的定时器必须是已经Start的：

- 步骤1 软件把需要启动的定时器TCB的地址，放在发给TRING的消息中发给TRING（实际上是发给POE指定的Group中）。
- 步骤2 TRING从POE获取到任务消息。
- 步骤3 TRING解析消息，获取到TCB的序号。
- 步骤4 TRING根据Stop命令中的TCB(x)序号以及TCB的起始地址，去读取TCB(x)。
- 步骤5 TRING把TCB(x)中的State位写为0b01(Created)。



步骤6 TRING把TCB(x)中的暂停位清0。

步骤7 TRING解挂定时器。

----结束

7.6.12 Delete 一个

全部由软件完成，Delete的定时器必须是已经Stop的：

步骤1 软件把TCB中的State域由0b01（Created）更新为0b00（Free）。

步骤2 软件把需要Delete的TCB的Nxt_idx刷新成原TCB FreeListPtr的指针。

步骤3 软件把TCB FreeListPtr的指针刷新成需要Delete的TCB地址。

----结束

7.6.13 TRING 挂接定时器操作

由TRING自己完成：

步骤1 计算挂接TCB参数：根据Wheel当前的时刻点，以及TCB(x)的Step_Hdr_No和Round_No，判断应该挂在哪个时刻点上，用 $(Round_No * All_Step_Hdr_No(wheel) + Step_Hdr_No) * T(Step_Hdr) + \text{当前时间点}$ 计算出到期时间；用Step_Hdr_No+当前的Step_Hdr_No得到需挂接的Step_Hdr位置。

步骤2 计算相关指针：读出挂接时刻点的Step_Hdr，如果为空（头尾指针都为NULL），则TCB(x)的Pre_idx要刷新为Step_Hdr的地址，否则TCB(x)的Pre_idx要刷新为Step_Hdr的Tail指示，根据Step_Hdr的Tail指示读出对应的TCB(t)，TCB(t)的Nxt_idx要刷新为AD(x)；两种情况下TCB(x)的Nxt_idx都要刷新为Null。

步骤3 刷新TCB：把Pre_idx，Nxt_idx，Wheel_No，Remain_rnd，State刷新到TCB(x)中。把AD(x)刷新到TCB(t)的Nxt_idx。

步骤4 刷新Step_Hdr：把挂接时刻点对应的Step_Hdr中的Tail刷新成TCB(x)的地址AD(x)，如果Step_Hdr中是空的，那么需要把Step_Hdr中的Head也刷新成AD(x)。

----结束

7.6.14 TRING 解挂定时器操作

由TRING自己完成：

步骤1 如果TCB(x)是挂在某Step_Hdr链条上的唯一TCB，则把Step_Hdr的Head和Tail都刷新为NULL。

步骤2 如果TCB(x)在Step_Hdr链条中间，把TCB(x)的Pre_idx写入后一个TCB(n)的Pre_idx，把TCB(x)的Nxt_idx写入前一个TCB(p)的Nxt_idx。

步骤3 否则如果TCB(x)在Step_Hdr第1个，把TCB(x)的Pre_idx写入后一个TCB(n)的Pre_idx，把TCB(x)的Nxt_idx刷新到Step_Hdr的Head。

步骤4 否则如果TCB(x)在Step_Hdr的最后1个，把NULL写入前一个TCB(p)的Nxt_idx，把TCB(x)的Pre_idx刷新到Step_Hdr的Tail。

----结束



7.6.15 处理定时器 WHEEL 操作

由TRING自己完成，在Ti时刻点上动作如下：

- 步骤1 用W_BASEAD(m) +i作为地址去读取Step_Hdr(i)。
- 步骤2 读出之后根据Head地址去读取第1个TCB，处理。
- 步骤3 根据第1个TCB中的Nxt_idx去读取第2个TCB，处理。
- 步骤4 重复步骤3，直到处理完最后一个TCB。
- 步骤5 等待基准时间结束。
- 步骤6 基准时间结束之后，到了下一个时间点Ti+1，回到1)继续处理第i+1个时间点上的任务。

----结束

7.6.16 处理定时器操作

由TRING自己完成，对具体的TCB(x)处理动作如下：

- 步骤1 读出TCB(x)之后，判断Exp_time是否到期，如果未到期（Exp_time>当前时刻点），不做任何处理，否则根据TCB中的参数，给POE发出消息（消息中就是本TCB的地址）或者上报完成中断。
- 步骤2 如果TCB(x)到期，且为周期性任务，则根据Step_Hdr_No和Round_No，以及当前的时刻点计算出要挂接的下一个时刻点，用(Round_No*All_Step_Hdr_No(wheel)+Step_Hdr_No)*T(Step_Hdr)+当前时间点计算出到期时间；用Step_Hdr_No+当前的Step_Hdr_No得到需挂接的Step_Hdr位置。
- 步骤3 在当前的Step_Hdr解挂（请参考[TRING解挂定时器操作](#)操作）。
- 步骤4 挂接到下一个Step_Hdr（请参考[TRING挂接定时器操作](#)操作）。

----结束

7.7 性能

一个Step_Hdr不超时能挂接最大的TCB数目。



图 7-5 step_hdr 最大挂接 TCB 数目分析

<p>处理定时器 (Step_Hdr第一个定时器) :</p> <p>1) 读DDR: 读Step_Hdr获取Head_TCB_idx 2) 读DDR: 用Head_TCB_idx作为地址读TCB 3) 写POE: 发POE消息 4) 写DDR: 刷新Step_Hdr的头尾指针 5) 写DDR: 刷新下一个TCB的pre_idx为NULL (需要刷新, 因为担心有stop的操作恰好是针对于下一个TCB的) 6) 读DDR: 读下一个挂接点的Step_Hdr 7) 写DDR: 刷新下一个挂接点Step_Hdr的头尾指针 8) 写DDR: 刷新下一个挂接点Step_Hdr的最后一个TCB的nxt_idx(如果下一个挂接点Step_Hdr为空则不需要) 9) 写DDR: 刷新当前TCB的pre_idx, nxt_idx</p> <p>$R1 + R2 + W4/W5 + R6 + W7/W8/W9 = 3R+2W$</p>	<p>处理定时器 (Step_Hdr中间一个定时器) :</p> <p>1) 读DDR: 读Step_Hdr获取Head_TCB_idx (不需要读, TCB的地址已经保存在了逻辑中) 2) 读DDR: 用Head_TCB_idx作为地址读TCB 3) 写POE: 发POE消息 4) 写DDR: 刷新Step_Hdr的头尾指针 5) 写DDR: 刷新下一个TCB的pre_idx为NULL (需要刷新, 因为担心有stop的操作恰好是针对于下一个TCB的) 6) 读DDR: 读下一个挂接点的Step_Hdr 7) 写DDR: 刷新下一个挂接点Step_Hdr的头尾指针 8) 写DDR: 刷新下一个挂接点Step_Hdr的最后一个TCB的nxt_idx(如果下一个挂接点Step_Hdr为空则不需要) 9) 写DDR: 刷新当前TCB的pre_idx, nxt_idx</p> <p>$R2 + W4/W5 + R6 + W7/W8/W9 = 2R+2W$</p>
<p>处理定时器 (Step_Hdr最后一个定时器) :</p> <p>1) 读DDR: 读Step_Hdr获取Head_TCB_idx (不需要读, TCB的地址已经保存在了逻辑中) 2) 读DDR: 用Head_TCB_idx作为地址读TCB 3) 写POE: 发POE消息 4) 写DDR: 刷新Step_Hdr的头尾指针 5) 写DDR: 刷新下一个TCB的pre_idx为NULL (需要刷新, 因为担心有stop的操作恰好是针对于下一个TCB的) 6) 读DDR: 读下一个挂接点的Step_Hdr 7) 写DDR: 刷新下一个挂接点Step_Hdr的头尾指针 8) 写DDR: 刷新下一个挂接点Step_Hdr的最后一个TCB的nxt_idx(如果下一个挂接点Step_Hdr为空则不需要) 9) 写DDR: 刷新当前TCB的pre_idx, nxt_idx</p> <p>$R2 + W4 + R6 + W7/W8/W9 = 2R+2W$</p>	<p>处理定时器 (Step_Hdr唯一的一个定时器) :</p> <p>1) 读DDR: 读Step_Hdr获取Head_TCB_idx 2) 读DDR: 用Head_TCB_idx作为地址读TCB 3) 写POE: 发POE消息 4) 写DDR: 刷新Step_Hdr的头尾指针 5) 写DDR: 刷新下一个TCB的pre_idx为NULL (需要刷新, 因为担心有stop的操作恰好是针对于下一个TCB的) 6) 读DDR: 读下一个挂接点的Step_Hdr 7) 写DDR: 刷新下一个挂接点Step_Hdr的头尾指针 8) 写DDR: 刷新下一个挂接点Step_Hdr的最后一个TCB的nxt_idx(如果下一个挂接点Step_Hdr为空则不需要) 9) 写DDR: 刷新当前TCB的pre_idx, nxt_idx</p> <p>$R2 + W4 + R6 + W7/W8/W9 = 2R+2W$</p>
<p>一个Step_Hdr上的TCB数目为n, 处理时间为$3R+2W+2R+2W+(n-2)*(2R+2W) = (2n+1)R+2nW \leq T$(wheel的最小精度)</p> <p>$n \leq (T-R)/(2(R+W))$</p> <p>假设R和W分别为150ns, 那么: T为100us时, $n \leq 167$; T为20us时, $n \leq 33$; T为10us时, $n \leq 16$</p>	

7.7.1 CPU 添加任务处理延迟分析

图 7-6 START 任务处理延迟分析

<p>对于START任务:</p> <p>1) 读DDR: 读TCB 2) 读DDR: 读要挂接的Step_Hdr 3) 读DDR: 读TCB(T)</p> <p>4) 写DDR: 写TCB 5) 写DDR: 写TCB(T) 6) 写DDR: 写Step_Hdr</p> <p>$R1/R2+R3+W4/W5/W6 = 2R+1W$</p>

图 7-7 STOP 任务处理延迟分析

对于STOP任务（挂在某个Step_Hdr的第一个）：

- 1) 读DDR：读TCB
- 2) 写DDR：写TCB（可以考虑不写）
- 3) 写DDR：写Step_Hdr
- 4) 写DDR：写TCB(T)

$$R1+W2/W3/W4 = 1R+1W$$

对于STOP任务（挂在某个Step_Hdr的中间一个）：

- 1) 读DDR：读TCB
- 2) 写DDR：写TCB（可以考虑不写）
- 3) 写DDR：写TCB(N)
- 4) 写DDR：写TCB(P)

$$R1+W2/W3/W4 = 1R+1W$$

对于STOP任务（挂在某个Step_Hdr的最后一个）：

- 1) 读DDR：读TCB
- 2) 写DDR：写TCB（可以考虑不写）
- 3) 写DDR：写Step_Hdr
- 4) 写DDR：写TCB(P)

$$R1+W2/W3/W4 = 1R+1W$$

对于STOP任务（挂在某个Step_Hdr的唯一一个）：

- 1) 读DDR：读TCB
- 2) 写DDR：写TCB（可以考虑不写）
- 3) 写DDR：写Step_Hdr

$$R1+W2/W3 = 1R+1W$$

7.7.2 发送 POE 消息吞吐量

图 7-8 ADD_POE 消息吞吐量

发poe消息需要的时间为T1(200ns)，假设OTD为m，假设恶劣情况（所有时间轮上的同时执行的Step_Hdr上都是最大不超时TCB数目符合正态分布概率）
 $167 * T1 * 32 * 0.317 / 2 / m < 100\mu s$
 $m > 167 * 32 * 0.317 / 2 * 200 / 100000 = 1.57 = 2$



7.8 DFX 应用指导

7.8.1 故障在线检测

7.8.1.1 TRING 配置异常

7.8.1.1.1 虚拟 wheel 与物理 wheel 不匹配

- 步骤1 检查寄存器WHEEL_MAP是否配置正常。
- 步骤2 检查SR_IOV_CTL相应VF使能是否正确打开。
- 步骤3 检查TCB数据结构里面的虚拟wheel_no域是否配置正确。
- 结束

7.8.1.1.2 虚拟 CPU 与物理 CPU 不匹配

- 步骤1 检查寄存器VCPU_MAP是否配置正常。
- 步骤2 检查SR_IOV_CTL相应VF使能是否正确打开。
- 步骤3 检查TCB数据结构里面的虚拟cpu_id号是否配置正确。
- 结束

7.8.2 维测统计

- 支持入口GET POE报文成功次数统计。
- 支持入口GET POE报文失败次数统计。
- 支持出口ADD POE报文次数统计。

7.8.3 典型定位案例

NA

7.9 注意事项

TRING处理POE任务与扫描step_hdr是互斥的，原因是操作同一个链表。

处理一个POE任务与扫描一个step_hdr之间采取SP调度。扫描step_hdr出现超时，轮子扫描为绝对优先级，直到追赶上wheel_cnt（时间计数）才会去处理POE任务。

当扫描step_hdr出现超时，必然是step_hdr挂接的TCB个数太多。扫描一个step_hdr的时间远大于处理一个POE任务时间。

当启动大量定时时长相同的周期性定时器时，扫描一个step_hdr的时间会是同样节点个数单次定时器的2倍（扫描+重新挂接），观察到TRING GET_POE的包率更低。

为了保证定时精度以及不反压get_poe，需要软件配合TRING保证挂接的均匀性。



需要软件配合的是：假设定时精度配置 $10\mu s$ ，有1000个10ms的定时任务，TCB配置中定时时长不要全都配置成10ms，需要每个错开一个定时精度（ $10\mu s$ ），降低挂接到同一个刻度。

周期性定时器、单次定时器都有此约束，当有大量定时周期相同的周期性定时器，推荐将step_hdr节点数配大，保证TCB能够散列挂接。

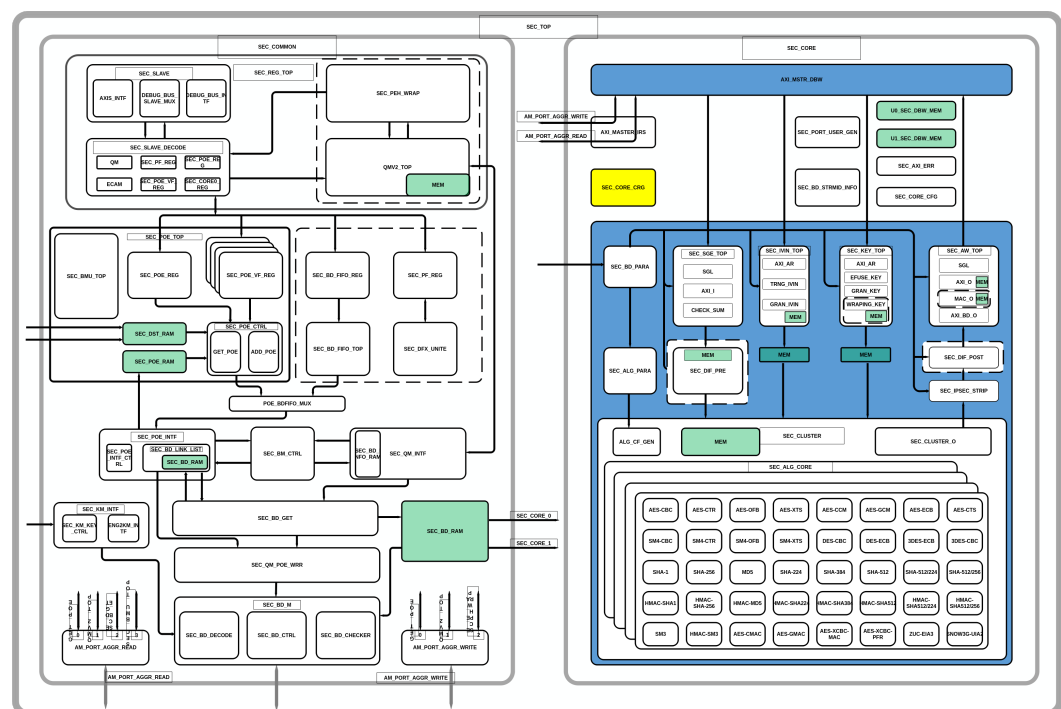
8 SEC

8.1 概述

SEC主要应用在认证、数据加解密等安全应用中，用以优化IPsec ESP、IPsec AH、TLS等协议的安全算法处理能力。

SEC模块实现了DES/3DES、AES、SM4、SM3、SHA-1、SHA-256、SHA-224、SHA-384、SHA-512、SHA-512/256、SHA-512/224、MD5算法，及基于SHA-1、SHA-256、SHA-224、SHA-384、SHA-512、SHA-512/256、SHA-512/224、MD5、SM3的HMAC运算、基于AES-CBC的MAC运算（AES-XCBC-MAC-96和AES-XCBC-PRF-128）、基于AES的CCM/GCM运算、基于AES的CMAC/GMAC运算、基于AES的XTS运算。DES/3DES支持ECB、CBC等2种工作模式，AES支持ECB、CBC、CTR、XTS等4种工作模式，SM4支持CBC、XTS等多种工作模式。

图 8-1 SEC 架构简图





8.2 关键术语

表8-1为与SEC相关的关键术语表。

表 8-1 SEC 加速器术语表

术语	描述
AAD	Additional authenticated data，附加认证数据。
AES	Advanced Encryption Standard，先进加密标准，一种加解密算法。
AH	Authentication Header，验证头，属于IPSEC的一种协议，用于为IP提供数据完整性，数据原始身份验证。
BD	Buffer Description，数据描述符。
CBC	Cipher Block Chaining，密码分组链接，加解密算法的一种工作模式。
CCM	Counter with CBC-MAC，计数及基于CBC模式的MAC，一种既有加解密又有认证的算法工作模式。
CMAC	Cipher-based Message Authentication Code，一种基于加密算法的认证算法。
CQ	完成队列，Completion Queue。
CQC	Completion Queue Context，完成队列上下文。
CTR	Counter，计数，加解密算法的一种工作模式。
DES	Data Encryption Standard，数据加密标准，一种通用加解密算法。
ECB	Electronic Code Book，电子密码本，加解密算法的一种工作模式。
ESP	Encapsulating Security Payload，封装安全协议，属于IPSEC的一种协议，可用于确保IP数据包的机密性，数据的完整性以及对数据源的身份验证。
EQ	Event Queue，中断事件队列。
EQC	Event Queue Context，中断事件队列上下文。
FIPS	Federal Information Processing Standards，（美国）联邦信息处理标准。
Flag	AES-CCM的IVIN组成元素，据其他参数计算得到。
GCM	Galois/Counter Mode，伽罗华/计数模式，一种既有加解密又有认证的算法工作模式。



术语	描述
GMAC	Galois Message Authentication Code，一种基于加密算法的认证算法。
hash	哈希，又称散列，一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。
HMAC	Hash-based Message Authentication Code，密钥相关的哈希运算消息认证码。
HUK	存储硬件特定对称性密钥（HUK-Hardware Unique Key）是随机产生，每个系统不同，仅用于密钥迁移。由HUK派生的密钥用于SST（trusted world Secure Storage可信世界安全存储）文件的加解密，可用于支持掉电存储，采用例如AES256算法。
ICV	Integrity check value，完整性校验值。
IPSEC	Internet Protocol Security，网络协议安全性。
MAC	Message Authentication Codes，消息认证码。
MD5	Message Digest, Version 5，报文摘要，版本5，一种认证算法。
NIST	National Institute of Standards and Technology，美国标准与技术研究所。
nonce	AES-CTR算法下用于组成IVIN。
SAA	Security Algorithm Accelerator，安全算法加速模块。
Salt	盐值，AES-CCM/AES-GCM算法下用于组成IVIN。
SEC	Security Engine，硬件安全加速模块（也即本模块），用以加强处理器的安全相关功能。
SGE	Scatter Gather Entry 聚散个体。每个SGE的大小为16Byte，描述一段连续的物理内存空间。
SGL	Scatter Gather List 聚散表，描述控制器与内存之间数据交互的一种机制。每个SGL包含多个SGE。
SHA	Security Hash Algorithm，安全散列函数，一种认证算法。
SP	Special Publications，特别报告书。
SQ	Submission Queue，提交队列。
SQC	Submission Queue Context，提交队列上下文。
XEX	XOR Encrypt XOR。
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing。

8.3 特性

SEC硬件加速IP有如下特性：



- 支持BD FIFO工作方式和POE工作方式（309a将SEC定义为两个设备：POE工作方式的SEC是非安全设备，即SEC_NS；BD FIFO工作方式的SEC是安全设备，即SEC_S）。
 - BD FIFO工作方式：仅用于安全世界软件通过SEC寄存器接口向SEC下发任务（一次配置一个BD的地址），多核并发时由安全世界软件完成调度。
 - POE工作方式：SEC和POE模块协同工作，仅用于非安全世界软件通过POE的队列向SEC下发任务。
 - BD FIFO和POE这两种工作方式可同时工作，安全世界软件使用BD FIFO工作方式，非安全世界软件使用POE工作方式。
- 支持BD描述符解析功能：
 - 支持通过解析BD获得流信息、控制信息、地址信息、长度信息等。
 - 一个BD描述符可支持一次认证和一次加解密操作，加解密和认证操作的数据范围可独立配置。
- 所有的数据类型都支持直接地址（即原始buf）访问：
 - 支持MAC、KEY、IVIN、待处理的数据或报文的地址字节对齐。
- 待处理的报文或者上下磁盘的数据还支持采用SGL访问接口：
 - 仅支持单SGL链。
 - 支持SGL数据结构指针64Byte对齐。
 - 支持SGE中的数据指针字节对齐。
 - 支持SGL数据结构中第一个SGE距离SGL头的偏移寄存器可配，至少支持配置如下3个值：32Byte、64Byte、128Byte。
 - 单个SGE的数据最小长度支持1Byte，最大长度需要支持到8MByte。
 - 单个SGL中的SGE的最少个数为1，最大个数支持到255。
 - SGL链中的SGL的最大总个数需要支持到256个。
- 待处理的报文还支持采用PPE访问接口：
 - 仅支持单PPE链。
 - 支持PPE数据结构指针64Byte对齐。
 - 支持PPE buff地址字节对齐。
- 支持IPSEC场景加解密和认证运算功能：
 - 支持单个BD处理的报文（IPSEC协议规定的所有域段，包含MAC部分以及跳过的报文头）长度上限是16MB-513B。
 - 支持的报文头（IP头+所有扩展头）长度上限是64KB-1B。
 - 支持本章节列出的所有加解密算法及其常用加解密模式（具体协议版本见本章节描述的算法部分）。
 - 支持本FS列出的所有认证算法，及与加解密算法的组合（除了AES-CCM、AES-GCM、AES-GMAC，具体协议版本见FS描述的算法部分）。
 - 加解密结果支持写出到新地址或原地址覆盖。
- IPSEC场景性能：
 - 平均加密后以太网报文长度为（原始报文512B，加密后大约512+70B，按照加密后报文计算带宽）512Byte时，AES_CBC 256+ HMAC_SHA 1/ HMAC_SHA256的吞吐率需 $\geq 15\text{Gbps}$ （3.66Mpps），延迟需 $\leq 10\mu\text{s}$ ，3DES/DES+HMAC_SHA 1/HMAC_SHA256吞吐率需 $\geq 4.7\text{Gbps}$ （1.12Mpps），延迟需 $\leq 20\mu\text{s}$ ，支持SM3，SM4，支持HMAC-SM3；SM3算法吞吐率需 $\geq 15\text{Gbps}$ （3.66Mpps）；SM3+SM4算法吞吐率需 $\geq 7\text{Gbps}$ （1.7Mpps）。



- 单流混合包长为 44:574:1500:300的比例为0.5:0.2:0.1:0.2的时候可以达到固定包长为300Byte的同等吞吐率，即AES_CBC 256+HMAC_SHA1的吞吐率需 $\geq 5.67\text{Gbps}$ (2.36Mpps)。
- IPsec单流性能 $\geq 4.32\text{Gbps}$ 。
- 支持所有常见的IPSEC加解密算法（本章节中列出的算法）一次加密加认证或解密加认证都能单独达到SEC.FS007.001~SEC.FS007.003中的性能。
- 支持SSL/TLS场景加解密和认证运算功能：
 - 支持单个BD处理的报文长度上限是（协议上限）18KB。
 - 支持报文头长度上限是64KB-1B。
 - 支持DES-CBC、3DES-CBC、AES-CBC-128/192/256加解密算法和AES-GCM-128/256等加解密认证混合算法。
 - 支持MD5、SHA1、SHA224、SHA256、SHA-384、SHA-512、SHA-512/256、SHA-512/224及其对应HMAC等认证算法。
- SSL/TLS场景性能：
 - SEC模块性能规格同IPSEC场景的SEC.FS007.001和SEC.FS007.002。
- 支持长数据hash / 流处理模式hash功能：
 - 支持MD5、SHA1、SHA224、SHA256、SHA-384、SHA-512、SHA-512/256、SHA-512/224、SM3、HMAC_MD5、HMAC_SHA1、HMAC_SHA224、HMAC_SHA256、HMAC_SHA-384、HMAC_SHA-512、HMAC_SHA-512/256、HMAC_SHA-512/224、HMAC_SM3算法。
 - 支持处理的数据长度最长到 $(2^{61} - 1)$ Byte。
 - 支持对一段长数据（报文），切分为1个首BD，若干个中间BD，1个尾BD，并分别配置给SEC，软件保证这些BD之间被SEC处理的顺序（需要原子保序）。
 - 首BD和中间BD，必须保证 Auth_Length > 0 且为算法Block的整数倍，MD5、SHA1、SHA224、SHA256、SM3及其对应HMAC算法，要求Auth_Length是64Byte的整数倍，SHA-384、SHA-512、SHA-512/256、SHA-512/224及其对应HMAC算法，要求Auth_Length是128Byte的整数倍。
 - 尾BD允许Auth_Length配置为任意长度，特别地Auth_Length=0，SEC也正常处理。
 - 1个首BD，1个尾BD，中间BD数量最小可以为0，最大数量，除需满足SEC.FS010.002和SEC.FS010.004要求外，无额外限制。
- 支持长数据加解密 / 流处理模式加解密功能：
 - 支持DES/3DES/AES/SM4的CBC模式、AES的CTR模式和GCM模式。
 - 支持处理的数据长度最长到SEC.FS013.001中各算法协议规定的上限（GCM模式支持的最大报文长度为16M-512B）。
 - 支持对一段长数据（报文），切分为1个首BD，若干个中间BD，1个尾BD，并分别配置给SEC，软件保证这些BD之间被SEC处理的顺序（需要原子保序）。
 - 首BD和中间BD，必须保证Cipher_Length > 0 且为算法Block的整数倍，DES/3DES算法，要求Cipher_Length是8Byte的整数倍，AES/SM4算法，要求Cipher_Length是16Byte的整数倍。
 - 对于CTR、GCM模式，尾BD允许Cipher_Length配置为 > 0 的任意长度；对于ECB、CBC模式尾BD对Cipher_Length配置要求同首BD和中间BD。
 - 1个首BD，1个尾BD，中间BD数量最小可以为0，最大数量，除需满足SEC.FS011.002、SEC.FS011.004、SEC.FS011.005要求外，无额外限制。



- 长数据 / 流处理模式场景性能：
 - 芯片加解密的性能规格同IPSEC场景下对应算法的性能规格（单个BD的数据长度一致时）。
 - 芯片hash/认证的性能规格同IPSEC场景下对应算法的性能规格（单个BD的数据长度一致时）。

说明

允许非流模式和流模式在同相同测试场景下性能差距为10%。

- 支持使用从SEC模块外部获取Key：
 - SEC无内置Key，只支持从外部获取Key。
 - 支持使用从DDR/Cache/Sram等存储设备读入的原始密钥。
 - 支持使用从E-Fuse获取的HUK。
 - 使用从DDR/Cache/Sram等存储设备读入的Key时，支持加解密和认证使用不同的Key。
 - 使用从E-Fuse等存储设备读入的KEY时，只支持加解密和认证使用相同的Key。
 - SEC从外部取回的所有Key，无论是安全还是非安全软件，都不能通过任何方式查询到。
- 支持使用IVIN：
 - 支持直接使用从外部读入的IVIN。
 - 支持从TRNG获取加密算法所需IVIN的随机数部分，并按照算法要求组成IVIN后使用，之后还会写出随机数。
 - Hash类算法支持使用内置的IVIN，各IVIN值遵照各算法协议规定。
 - 加解密和认证算法支持使用各自的IVIN。
 - 支持从外部获取加解密算法所需IVIN的随机数部分，并按照算法要求组成IVIN后使用。
- 支持ICV Check功能：
 - 支持对计算出的MAC值与MAC_Addr处所存储的MAC值进行比较，比较结果可以回写到BD里。
- IPSEC支持IPv6 UDP Checksum功能（解密方向不支持带扩展头场景），加密方向由硬件计算，解密方向由硬件校验。
- 支持TLS场景下对待加密数据做Padding：
 - 支持不做Padding。
 - 支持做PKCS5 Padding。
 - 软件指示Padding数据长度和内容。
- 支持AES加、解密运算，符合NIST FIPS-197标准：
 - AES密钥长度支持128、192、256 bit。
 - AES支持ECB、CBC和CTR工作模式，工作模式符合NIST的Special800-38A标准。
 - 支持基于AES的AES_XCBC_MAC96算法，符合RFC3566协议标准。
 - 支持基于AES的AES_CMAC算法，符合RFC 4493协议标准（Special800-38B标准）。
 - 支持基于AES的AES_CCM算法，即Counter with CBC_MAC算法，用于IPSEC，符合RFC 4309协议标准。



- 支持基于AES的AES_GCM算法，即Counter with GMAC算法，符合RFC4106协议标准。
- 支持基于AES的AES-XCBC-PRF-128算法，用于产生随机数，符合RFC4434协议标准（虽然协议中对KEY的长度支持任意bit，但因实际场景仅需要KEY的长度为128bits，故SEC固定支持KEY的长度为128bits）。
- 支持AES-XTS算法，符合IEEE P1619协议标准。
- 支持基于AES的AES_GMAC算法，即Galois Message Authentication Code算法，符合RFC4543协议标准。
- 支持DES/3DES加、解密运算：
 - 3DES支持3个密钥，也支持2个密钥。
 - DES/3DES支持ECB、CBC这两种工作模式，工作模式符合FIPS-81标准。
 - DES-CBC符合RFC2405协议标准。
 - 3DES-CBC符合RFC2451协议标准。
- 支持NULL算法，符合RFC2410协议标准。
- 支持SM4算法，遵照国家密码管理局于2012年3月21日发布的版本：GM/T 0002-2012。
- 支持SHA-1数据摘要算法，符合FIPS 180-4标准要求；SHA-1支持160 bits的数据摘要。
- 支持SHA-224、SHA-256、SHA-384、SHA512、SHA-512/256、SHA-512/224数据摘要算法，符合FIPS 180-4标准要求；SHA-224支持224 bits的数据摘要，SHA-256支持256bits的数据摘要，SHA-384支持384 bits的数据摘要，SHA-512支持512bits的数据摘要，SHA-512/256支持256bits的数据摘要，SHA-512/224支持224bits的数据摘要。
- 支持MD5数据摘要算法，符合RFC 1321标准的要求；MD5支持128 bits的数据摘要。
- 支持SM3算法，遵照国家密码管理局于2012年3月21日发布的版本：GM/T 0004-2012。
- 支持HMAC，符合RFC2104标准，HMAC可选择SHA-1、SHA-224、SHA-256、SHA-384、SHA-512、SHA-512/256、SHA-512/224、MD5、SM3中任何一种HASH算法。
- SEC大小端支持说明：
 - SEC寄存器只支持小端模式。
 - SEC PD（跟POE交互的数据结构），只支持小端模式。
 - SEC BD，支持小端模式、32bit大端模式（在32bit内进行字节序翻转，32bit之间的顺序保持不变）和64bit大端模式（在64bit内进行字节序翻转，64bit之间的顺序保持不变）三种模式寄存器可配置；但SGL数据结构固定支持小端模式。
 - 大小端配置默认为小端模式。
 - SEC IVIN、KEY、MAC（指纹信息）、数据明文、数据密文均仅支持大端模式。
- 支持动态功耗控制。
 - 支持各算法核在没有处理数据时时钟被硬件自动关断，此功能有独立使能位。
- 支持密钥派生算法PBKDF2，符合RFC2898协议标准。



- 支持的Salt长度为8B~1023B。
- 支持的PassWord的长度为6B~64B (HMAC-SHA256、HMAC-SHA224、HMAC-SM3可以到64B, HMAC-SHA-384、HMAC-SHA512、HMAC-SHA-512/256、HMAC-SHA-512/224可以到128B), 当前产品应用范围为6B~32B。
- 支持DK的长度为8B~512B, 典型值为32B和64B。
- 支持hash算法的迭代次数为1次~10M次, 典型值为10000次。
- 支持的算法为HMAC-SHA256、HMAC-SHA224、HMAC-SHA-384、HMAC-SHA512、HMAC-SHA-512/256、HMAC-SHA-512/224、HMAC-SM3。
- 只有第3个通道支持PBKDF2算法, 需要SEC支持PBKDF2算法必须使能第3个通道。
- PBKDF2性能。
 - 仅HMAC-SHA256有性能要求, 其他算法不做性能要求。
 - 在典型输入和输出参数下 (PassWord长度为32B, Salt长度为32B, DK长度为32B, 迭代10000次), HMAC-SHA256的吞吐率 $\geq 0.5\text{Kpps}$, 延迟 $\leq 2\text{ms}$ 。
 - 当PBKDF2任务和加解密认证运算任务同时存在时, 允许加解密认证运算吞吐率相比于无PBKDF2任务时下降不超过35%。
 - 注: 当PBKDF2任务和加解密认证运算任务同时存在时, 当PBKDF2任务个数大于2时会产生反压, 无法GET POE。
- 支持通过接口从E-Fuse获取HUK。
- 支持通过E-Fuse禁止SM3、SM4、DES、3DES、SHA1、MD5算法的使用。
 - 支持通过烧写E-Fuse相关bit位禁止SM3、SM4算法的使用, 该禁止不可撤销: 即一旦烧写成功后该芯片就再也不能使用SM3、SM4算法。
 - 支持通过烧写E-Fuse相关bit位禁止DES、3DES算法的使用, 该禁止不可撤销: 即一旦烧写成功后该芯片就再也不能使用DES、3DES算法。
 - 支持通过烧写E-Fuse相关bit位禁止SHA1、MD5算法的使用, 该禁止不可撤销: 即一旦烧写成功后该芯片就再也不能使用SHA1、MD5算法。
- 支持POE相关功能。
 - 支持POE接口输出PF中断。
 - 支持多模之间隔离。
 - 支持芯片内所有模块的统一的设备和功能标识号 (设备号PF_ID, 功能号VF_ID), 供软硬件之间以及加速器模块之间相互识别。
 - 支持芯片提供系统缺省标识号, 并且支持软件配置修改。
- 支持安全任务输出安全中断。
- 向KM申请2种密钥。
 - 安全软件控制SEC向KM申请Wrapping KEY、STORAGE_KEY 2种类型的KEY, 每种KEY的位宽都是256bit。
 - SEC支持软件控制向KM发起加载功能。
 - SEC使用STORAGE_KEY具有严格的安全权限控制: KM授权且只能在安全中使用。
- 支持使用Wrapping算法加密后的KEY。
 - SEC支持Wrapping KEY、STORAGE_KEY使用GCM解密。
 - Wrapping KEY、STORAGE_KEY用于解密DDR中的KEY 密文。



- BD中选中Wrapping KEY即可使用该模式。
 - 安全任务下，BD中选中STORAGE_KEY才可使用该模式。
 - 使用Wrapping KEY、STORAGE_KEY的性能不低于相同场景下使用明文KEY性能（五条流以上）。
- 支持ECC功能。
 - SEC支持ECC 2bit检错1bit纠错。
 - SEC支持ECC 单bit出错后纠正回写到SRAM中。
 - SEC支持记录ECC错误的地址及现场，并对1bit ECC错误进行计数。
- 支持Memory相关属性。
 - SEC支持安全与非安全访问（默认非安全，可配置安全地址段）。
 - SEC支持片内SRAM初始化后，SRAM内容在非上电复位过程中做数据保持。
- 支持加速器相关属性。
 - 支持加速器之间的直接通信，减少与CORE的交互。
 - 支持HAC之间的中断不合并，每个HAC的异常和正常中断分开。
- 支持单BD处理延迟统计。
 - 支持单BD处理最大延迟统计。
 - 支持单BD处理平均延迟统计。
 - 支持单BD处理最小延迟统计。
 - 支持统计类寄存器缺省为读清。
- 支持Debug Bus接口。
- 支持SEC内部存储BD的memory的可视化，通过寄存器可以查看SEC内部存储BD的memory中的数据。
- 支持AXI总线异常检测，当SEC在处理报文、SGL链、IVIN、KEY、MAC等辅助数据发生总线错误或者回写报文数据发生总线错误时，会上报对应中断。

8.4 遵循协议

AES加、解密运算，符合NIST FIPS-197标准；

AES支持ECB、CBC和CTR工作模式，工作模式符合NIST的SP 800-38A标准；

AES-CBC with 128-bit key算法，符合RFC3602协议标准；

AES-CTR算法，符合RFC3686协议标准；

AES_XCBC_MAC96算法，符合RFC3566协议标准；

AES_CCM算法，即Counter with CBC_MAC算法，符合RFC4309协议标准；

AES_GCM算法，即Counter with GMAC算法，符合RFC4106协议标准；

AES_GMAC算法，即Galois Message Authentication Code算法，符合RFC4543协议标准；

AES-XCBC-PRF-128算法，符合RFC4434协议标准；

AES-XTS是基于IEEE P1619；

DES/3DES加、解密运算，符合FIPS 46-3标准；



DES/3DES支持ECB、CBC几种工作模式，工作模式符合FIPS-81标准；

DES-CBC算法，符合RFC2405协议标准；

3DES-CBC算法，符合RFC2451协议标准；

NULL算法，符合RFC2410协议标准；

SM4算法，遵照国家密码管理局于2012年3月21日发布的版本：GM/T 0002-2012；

SHA-1报文摘要算法，符合FIPS 180-2标准要求；SHA-1支持160 bits的报文摘要；

SHA-224、SHA-256、SHA-384、SHA512、SHA-512/256、SHA-512/224数据摘要算法，符合FIPS 180-4标准要求；SHA-224支持224 bits的数据摘要, SHA-256 支持256bits的数据摘要，SHA-384支持384 bits的数据摘要，SHA-512支持512bits的数据摘要，SHA-512/256支持256bits的数据摘要，SHA-512/224支持224bits的数据摘要；

MD5报文摘要算法，符合RFC 1321标准的要求；MD5支持128 bits的报文摘要；

HMAC，符合RFC2104标准，HMAC可选择SHA-1、SHA-224、SHA-256、SHA-384、SHA512、SHA-512/256、SHA-512/224、MD5、SM3中任意一种HASH算法；

SM3算法，遵照国家密码管理局于2012年3月21日发布的版本：GM/T 0004-2012；

8.5 功能描述

8.5.1 IPsec 机密性和完整性协议处理

IPSEC下报文安全封装协议为ESP、AH协议，ESP协议提供机密性和完整性保护，AH协议仅提供完整性保护，不过AH协议对报文完整性保护的的范围大于ESP协议的。

表 8-2 IPv4 Head

-	IPv4 Head															-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	3								2								1								0							
-	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0			
1	Version			IHL					Type of Services								Total length															
2	Identification															Flags			Fragment offset													
3	TTL								Protocol								Header checksum															
4	SrcIPAddr																															
5	DstIPAddr																															

 说明

这里的Header checksum是对这20Byte进行校验和计算，被校验的20Byte中Header checksum域值填全零参与计算。

表 8-3 IPv6 Head

-	IPv6 Head																-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	3								2								-	1								0											
-	3	3	-	2	2	2	2	2	2	2	2	2	1	1	1	1	-	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
1	Version =6				category								-	flowId																							
2	Payload Length																-	Next Head								maxhop											
3	-																																				
4	-																																				
5	-																																				
6	-																																				
7	-	DstIPAddr																																			
8	-																																				
9	-																																				
10	-																																				

上面的数据结构没有画出IPv6的扩展头（extension header），扩展头紧跟着DstIPAddr。Hi309a里IPv6扩展头支持：

- 分片扩展头（Next Header=8' d44）（解密方向SEC不支持）
- 授权（AH）扩展头（Next Header=8' d51）
- 封装安全有效载荷（ESP）扩展头（Next Header=8' d50）
- 上层扩展头TCP头（Next Header=8' d6），上层扩展头UDP头（Next Header=8' d17），上层扩展头SCTP头（Next Header=8' d132），上层扩展头ICMP头（Next Header=8' d58），上层扩展头Iplnlp（Next Header=8' d4）



图 8-2 ESP Head

ESP Head																																							
3								2								1								0															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
SPI																																							
Sequence Number																																							
IV																																							
说明																																							
SPI								接收方根据报文中的SPI查找SA数据库，找到对应的SA。SPI值0被保留来表明“没有安全关联存在”																															
Sequence Number								从1开始的32位单增序列号，不允许重复，唯一地标识了每一个发送数据包，为安全关联提供反重播保护。接收端校验序列号为该字段值的数据包是否已经被接收过，若校验为已接收过，则拒收该数据包																															
IV								某些加密算法需要使用IV，这种特殊加密算法必须定义IV的位置，IV是受保护数据的第一个8位组。对IV进行验证但不进行加密。(IV值的长度不确定)																															

图 8-3 ESP TAIL

ESP TAIL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
----------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

这里的TCP/UDP并不仅仅占一个word，它的意思是：若Auth Header中的Next Header=8’ d6即是TCP，那么这里“TCP/UDP”表示的是TCP报文头；若Auth Header中的Next Header=8’ d17即是UDP，那么这里“TCP/UDP”表示的是UDP报文头；

IPv6的传输、隧道格式类似于IPv4时，就不再一一画出；

下面的表格都是如此。

图 8-6 IPv4 ESP 隧道模式

IPv4 ESP Tunnel Mode																																
3								2								1								0								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Version				IHL				Differentiated Services								Total length																New IP Header
Identification																Flags		Fragment offset														
TTL								Protocol								Header checksum																
SrcIPAddr																																
DstIPAddr																																ESP Header
SPI																																
Sequence Number																																
IV																																
Version				IHL				Differentiated Services								Total length																IP Header
Identification																Flags		Fragment offset														
TTL								Protocol								Header checksum																
SrcIPAddr																																
DstIPAddr																																
TCP/UDP																																
Data																																
Padding																																ESP Tail
Padding																Padding Length								Next Header								
ESP Auth Data																																
说明																																
相比传输模式而言新增加了20字节的IP头																																

图 8-7 IPv4 ESP 隧道 Nat 模式

IPv4 ESP Tunnel NAT Mode																																	
3								2								1								0									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Version				IHL				Differentiated Services								Total length																New IP Header	
Identification								Flags								Fragment offset																	
TTL								Protocol								Header checksum																	
SrcIPAddr																																	
DstIPAddr																																	
SrcPort																DstPort (4500)																UDP Header	
Length																Checksum																	
SPI																																	
Sequence Number																																	
IV																																	
Version				IHL				Differentiated Services								Total length																IP Header	
Identification								Flags								Fragment offset																	
TTL								Protocol								Header checksum																	
SrcIPAddr																																	
DstIPAddr																																	
TCP/UDP																																	
Data																																	
Padding																																ESP Tail	
Padding																Padding Length								Next Header									
ESP Auth Data																																	
说明																																	
相比ESP Tunnel而言, NAT模式上新增加了UDP头部并且UDP目的端口号为4500。																																	

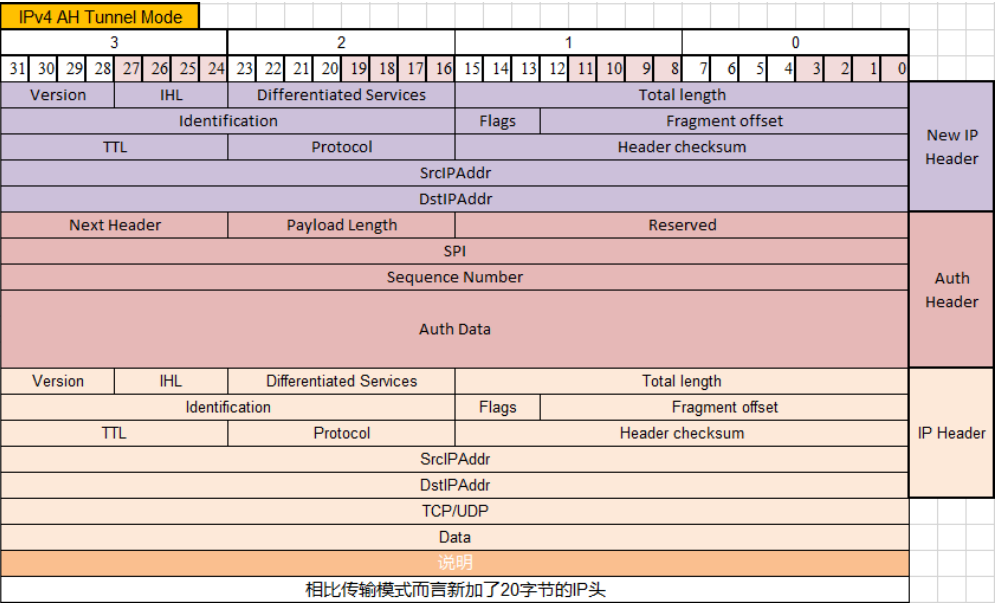
只有IPv4 和IPv6的ESP隧道模式支持。

图 8-8 IPv4 AH 传输模式

IPv4 AH Transfer Mode																															
3								2								1								0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Version				IHL				Differentiated Services								Total length								IP Header							
Identification								Flags				Fragment offset																			
TTL				Protocol				Header checksum																							
SrcIPAddr																															
DstIPAddr																															
Next Header				Payload Length				Reserved								Auth Header															
SPI																															
Sequence Number																															
Auth Data																															
TCP/UDP																															
Data																															
说明																															
Next Header		描述在AH认证头的下一个载荷的类型。例如，值为6代表TCP。																													
Payload Length		AH报头长度，以32bit为单位计算结果再减2。典型值为4。																													
Reserved		用于以后的扩展。发送方必须置该位为0，并且接受者应该忽略该字段。																													
SPI		接收方根据报文中的SPI查找SA数据库，找到对应的SA。SPI值0被保留来表明“没有安全关联存在”从1开始的32位单增序列号，不允许重复，唯一地标识了每一个发送数据包，为安全关联提供反重播保护。接收端校验序列号为该字段值的数据包是否已经被接收过，若校验为已接收过，则拒收该数据包																													
Sequence Number		是一个长度可变的域，包含完整性检查值（ICV）。长度为32bit的整数倍，通常为96Bit。接收端接收数据包后，首先执行带密钥的hash计算，再与发送端所计算的该字段值比较，若两者相等，表示数据完整，若在传输过程中数据遭修改，两个计算结果不一致，则丢弃该数据包。																													
Auth Data																															



图 8-9 IPv4 AH 隧道模式



8.5.1.1 IPsec ESP 协议处理

基于BD结构灵活进行配置，软硬件协同可以完成IPsec ESP机密性和完整性处理。

图8-10、图8-11是在上行方向机密性和完整性都需要进行处理的示意图，前者是传输模式，后者是隧道模式。对于SEC来说，虽然两者的包结构有所不同，但处理流程是相同的。

在上行方向，对报文先进行加密，后进行认证。软件通过BD指示需要采用的机密性算法、完整性算法、机密性密钥、完整性密钥等信息，SEC通过解析BD得到需要加密数据的起始地址、长度信息，需要认证数据的起始地址、长度信息，然后读入相应数据完成加密运算、认证运算，并将处理后的数据（结果）输出到指定位置。下图中红色箭头区间表示认证范围，蓝色箭头区间表示加密范围。

图 8-10 IPsec ESP 传输模式上行方向 SEC 处理

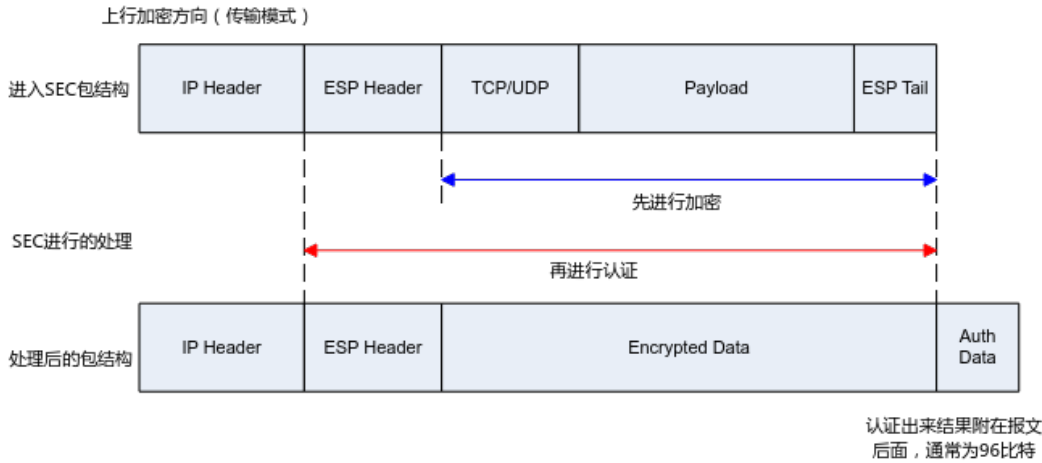
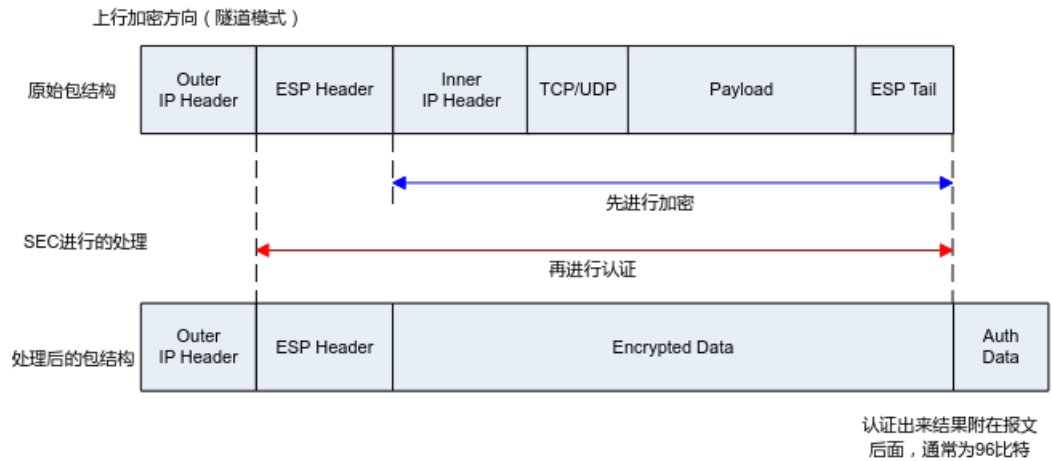


图 8-11 IPsec ESP 隧道模式上行方向 SEC 处理



如图8-12和图8-13所示是在下行方向机密性和完整性都需要进行处理的示意图，前者是传输模式，后者是隧道模式。对于SEC来说，虽然两者的包结构有所不同，但处理流程是相同的。

在下行方向，对报文先进行认证，后进行解密。软件通过BD指示需要采用的机密性算法、完整性算法、机密性密钥、完整性密钥等信息，SEC通过解析BD得到需要认证数据的起始地址、长度信息，需要解密数据的起始地址、长度信息，然后读入相应数据完成认证运算、解密运算，并将处理后的数据（结果）输出到指定位置（DE=0时，payload部分原址回写）。SEC会比较认证计算出的ICV值，与对端带过来的ICV值是否一致，比较结果通过BD回写。如图8-12所示中红色箭头区间表示认证范围，蓝色箭头区间表示解密范围。

图 8-12 IPsec ESP 传输模式下行方向 SEC 处理

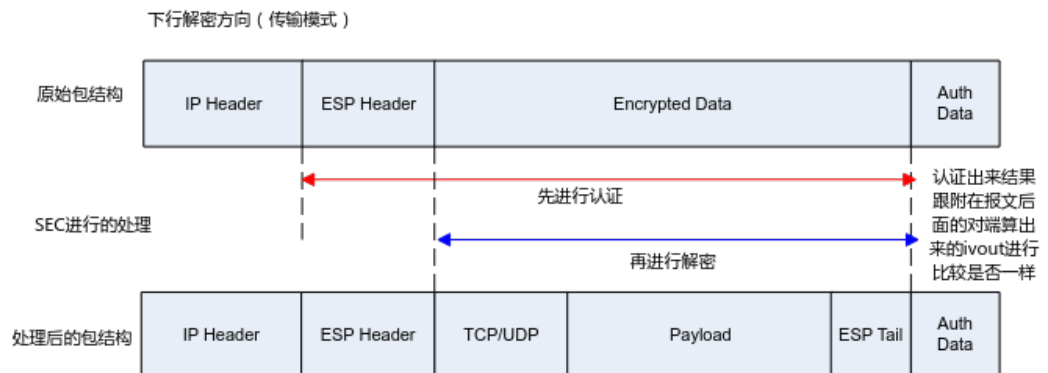
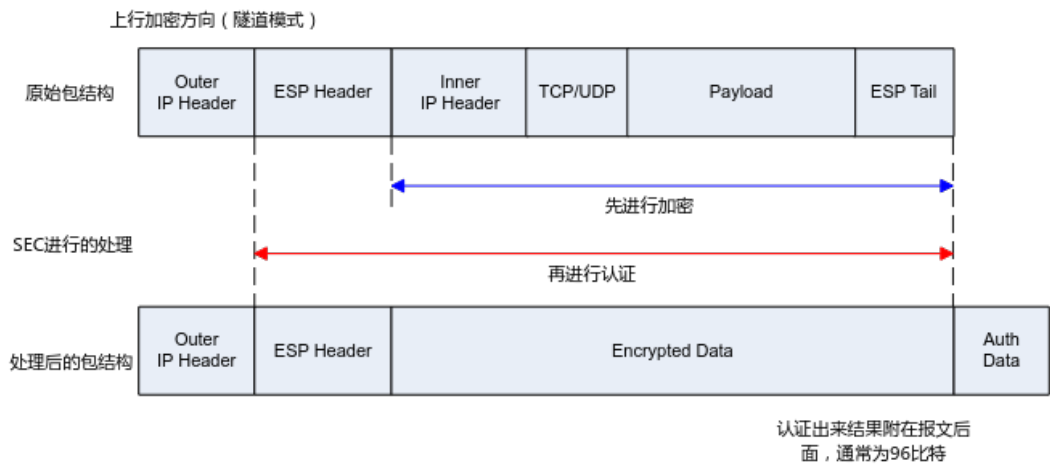


图 8-13 IPsec ESP 隧道模式下行方向 SEC 处理



8.5.1.2 IPsec AH 协议处理

基于BD结构灵活进行配置，可以完成IPsec AH完整性处理。

如图8-14和图8-15所示是在IPsec AH协议中在上行方向完整性协议的一个示意图，两者的区别在于一个是传输模式，一个是隧道模式，进入SEC时，原始包结构有所不同。从两个图的对比可以看到，需要认证的数据长度是不一样的，但对于SEC来说，处理的流程是一样的，都是对某段数据进行认证操作。

在上行方向，软件通过BD指示需要采用的完整性算法，完整性密钥等信息，然后将需要认证的数据的地址以及认证报文总长度（图8-14和图8-15中红色箭头对应报文）传给SEC，SEC对其完成认证操作，计算出一个ICV值，并将ICV值写在AH Header中的ICV域，通常为96bit（也可以选择其他长度，具体由完整性算法配置决定）。

在实际使用中，IP Header中的部分域（包括服务类型、存活时间等）在IP包不断转发的过程中不断变更，这些可变域和ICV在计算HASH时需要以0来代替。这些可变域和ICV域需要在计算ICV前由软件初始化为0。

图 8-14 IPsec AH 传输模式上行方向 SEC 处理

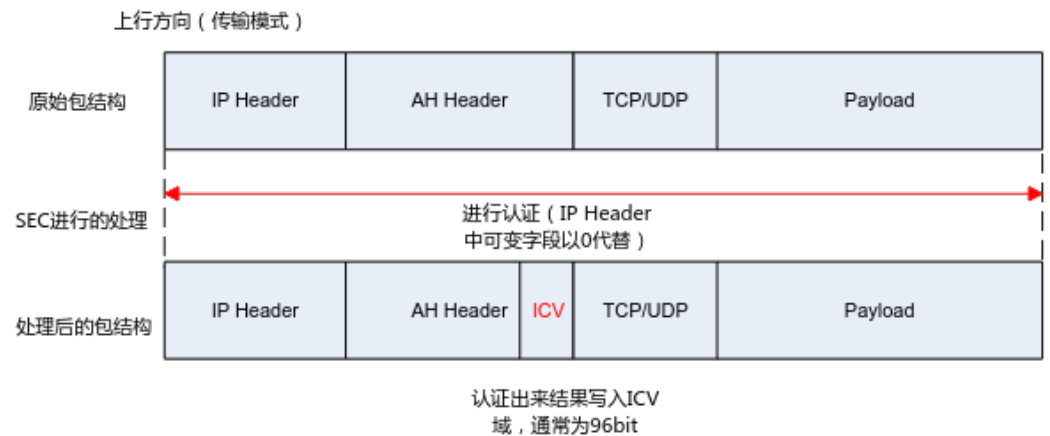
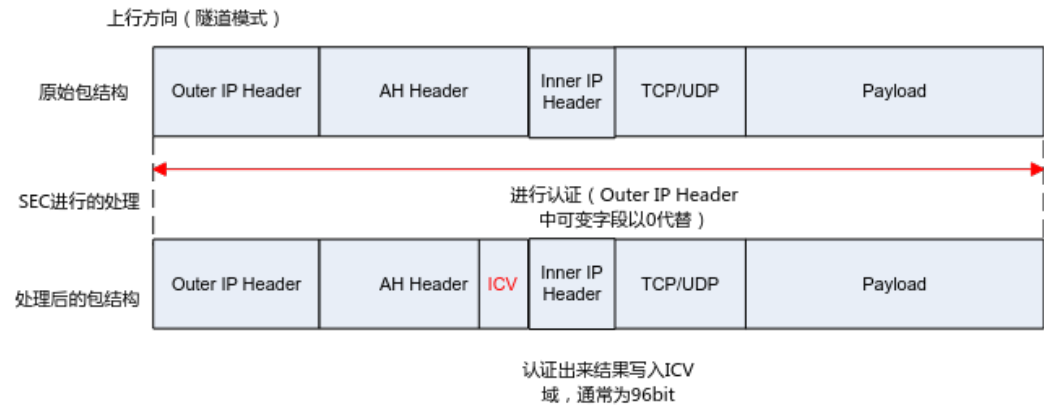


图 8-15 IPsec AH 隧道模式上行方向 SEC 处理



如图8-16和图8-17所示是指在IPSEC AH协议中在下行方向完整性协议的示意图，前者是传输模式，后者是隧道模式，两者进行认证数据的范围有区别。

在下行方向，通过BD中指示需要采用的完整性算法，完整性密钥等信息，然后将需要认证的数据的地址以及认证报文总长度（图8-16和图8-17中红色箭头对应报文）传给SEC，SEC对其完成认证操作，计算出一个ICV值，并将处理后的数据（结果）输出到指定位置；SEC还会将计算的ICV值与对端带过来的ICV值（AH Header中的ICV域）进行比较，比较结果通过BD回写反馈。

在实际使用中，IP Header中的部分域（包括服务类型、存活时间等）在IP包不断转发的过程中不断变更，这些可变域和ICV在算HASH时需要以0来替代。这些可变域和ICV域在计算ICV前由软件初始化为0（真正的ICV软件需要搬到别处，并将新的地址做为MAC地址通过SEC BD传给SEC；IP Header中的可变域软件需要搬到别处，SEC处理完毕后软件再搬回）。

图 8-16 IPsec AH 传输模式下行方向 SEC 处理

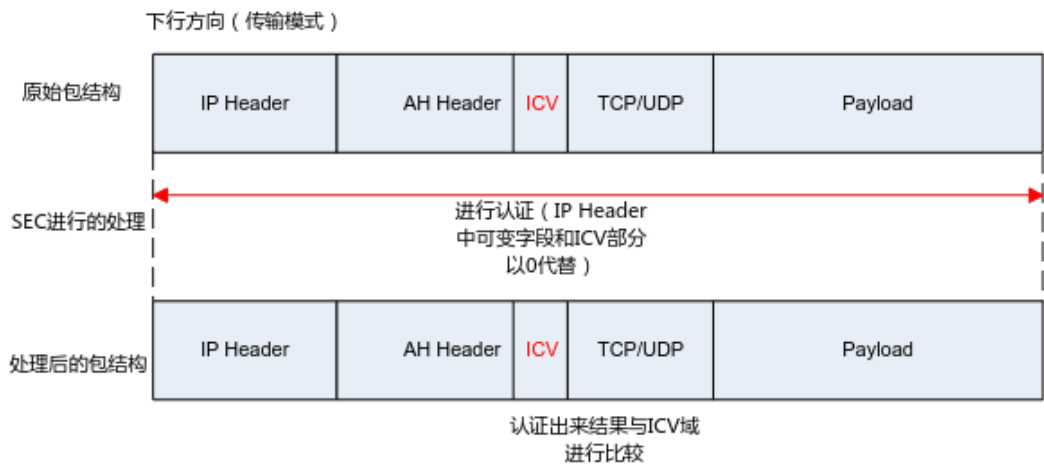
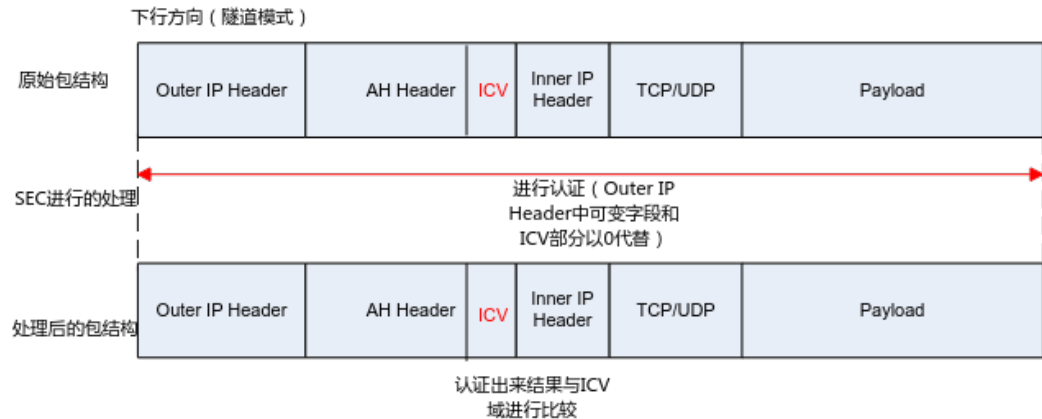


图 8-17 IPsec AH 隧道模式下行方向 SEC 处理

**须知**

对应同一个报文既需要做IPsec ESP认证又需要做IPsec AH认证的情况，SEC必须分为2个BD来处理，且这两个BD间由软件来进行原子保序。

8.5.2 OpenSSL/TLS1.2 机密性和完整性协议功能

8.5.2.1 OpenSSL/TLS1.2/DTLS 报文格式

讨论场景时，OpenSSL和TLS场景等同。

报文头长度：TLS场景下为5B；DTLS场景下为13B（DTLS除了头部跟TLS不同以外，其他都等同于TLS）；

IV长度：DES/TDES算法是8B；AES算法CBC模式是16B，GCM，CCM模式是8B。

8.5.2.2 OpenSSL/TLS 软件配置

计算场景下需要支持OpenSSL/TLS协议机密性以及完整性协议功能，基于SEC的BD结构灵活进行配置，可以完成该功能。

图8-18为OpenSSL/TLS协议中SEC下行方向单加密处理：

SEC可据软件配置，添加Padding、Pad_Len，修改Len域。

SEC需要对Payload、Padding、Pad_Len部分进行加密，并在输出时在Len域和加密后数据间插入IV。

图 8-18 OpenSSL/TLS 下行方向 SEC CBC 算法单加密处理



图 8-19 OpenSSL/TLS 下行方向 SEC CBC 算法单加密处理后插入 IV



TLS_Len_Update域段配置为1；

Cipher_Offset域段配置不小于报文头长度；

Cipher_Len域段：

1. 当Cipher_Pad_Type配置为0x0时，Cipher_Len域段包含payload + pad，需根据算法协议block对齐。
2. 当Cipher_Pad_Type配置为0x1/0x2：Cipher_Len域段包含payload，源地址空间大小和目的地址空间大小保持一致。
3. 当Cipher_Pad_Type配置为0x3：Cipher_Len域段包含payload,软件需Cipher_Len + Cipher_Pad_Len + 1满足协议的block对齐。

须知

SEC仅支持CBC、CCM、GCM算法，CCM/GCM的约束见下方描述。

图8-20为OpenSSL/TLS上行方向SEC CBC算法单解密处理：

SEC需要对EncData解密解出Payload、Padding、Pad_Len部分，并在输出时剥除在Len域和Payload数据间的IV；并可据软件配置，进一步剥除Padding、Pad_Len，修改Len域。

图 8-20 OpenSSL/TLS 上行方向 SEC CBC 单解密处理

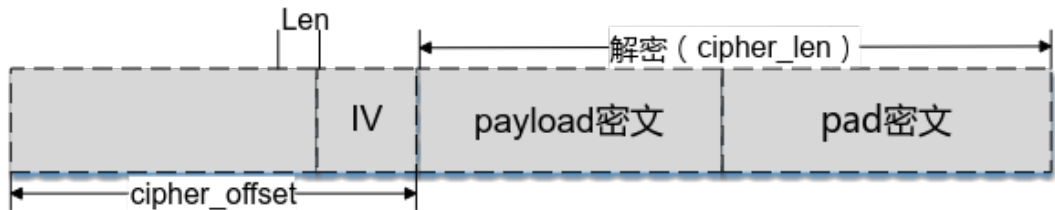
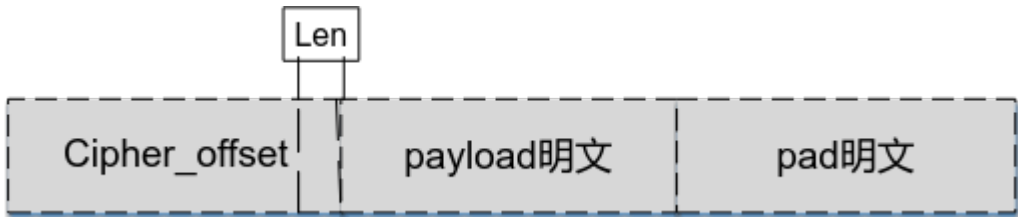


图 8-21 OpenSSL/TLS 上行方向 SEC CBC 单解密处理后去除 IV



TLS_Len_Update域段配置为1；

Cipher_Offset域段配置不小于报文头长度 + IV长度；

Cipher_Len域段包含payload密文 + pad密文，需根据算法协议block对齐；

SEC解密完pad_len通过BD回写，软件更新报文头的Len域（ $Len(新) = Len(原) - Tls_Pad_Len_I - 1$ ）；

图8-22为OPENSSL/TLS上行/下行方向SEC认证/认证校验处理：

SEC需要对Sequence Number、Type、Version、Len、Payload部分做认证，算出的MAC值附在Payload之后。需要特别注意的是，Sequence Number是由软件通过BD配置传递给SEC的，计算认证时是插在Type域前面，计算完成后并不需要（也不能随报文）输出。

SEC在认证方向将算出的MAC值写到MAC_ADDR，认证校验（也叫解认证）方向从MAC_ADDR读取MAC值与SEC算出的MAC比较，比对通过则回写BD中ICV域为1，不通过则回写为2。

图 8-22 OpenSSL/TLS 上行/下行方向 SEC 认证/认证校验处理



BD中的SN域段填写Sequence Number，输出的报文不包含Sequence Number；

Auth_Len域段大于报文头长度；

须知

不支持AES_GMAC算法。不支持Cipher域段配置为3，单认证下Cipher配置为0即可实现搬移。

如图8-23所示为OpenSSL/TLS下行方向SEC先做认证，再做加密处理：

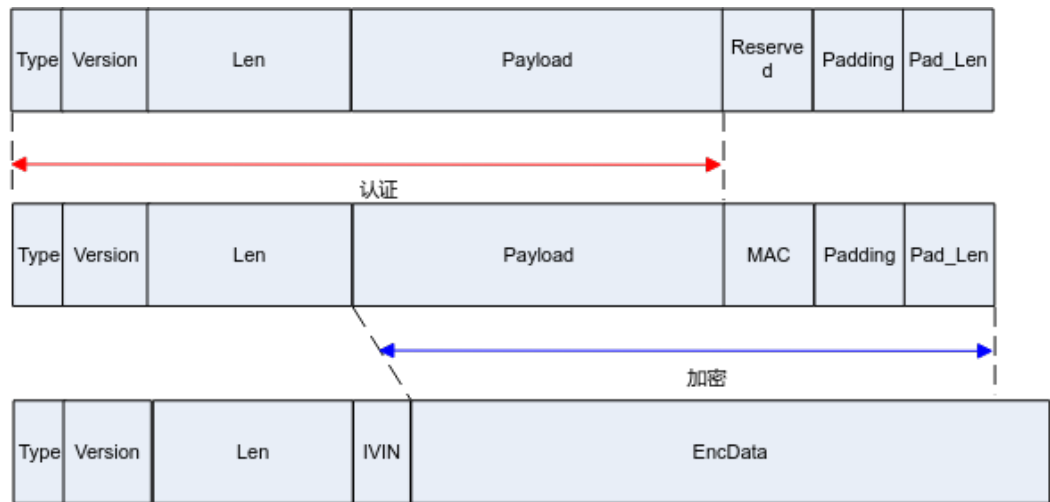
SEC需要对Sequence Number、Type、Version、Len、Payload部分做认证，算出的MAC值附在Payload之后；

SEC可据软件配置，添加Padding、Pad_Len，修改Len域。

SEC需要对Payload、MAC、Padding、Pad_Len部分进行加密，并在输出时在Len域和加密后数据间插入IV。

SEC可据软件配置，在加密后修改Len域为 $Len(Payload) + Len(IV) + Len(MAC) + Len(Padding) + 1$ 。

图 8-23 OpenSSL/TLS 下行方向 SEC 认证+CBC 加密处理



SEQ域段配置为1；

TLS_Len_Update域段配置为1；

Cipher域段配置为1；

Auth域段配置为1；

Cipher_Len域段：

- Cipher_Pad_Type配置为4'h0: Cipher_Len域段包含payload + mac+pad，需根据算法协议block对齐。
- Cipher_Pad_Type配置为4'h1/4'h2: Cipher_Len域段包含payload，源地址空间大小和目的地址空间大小保持一致。
- Cipher_Pad_Type配置为4'h3: Cipher_Len域段包含payload + mac,软件需Cipher_Len + Cipher_Pad_Len + 1满足协议的block对齐。
- Cipher_Offset域段配置不小于报文头长度。
- Auth_Len配置为报文头长度+ payload。
- Auth_Offset配置为Cipher_Offset - 报文头长度。

图8-24为OpenSSL/TLS上行方向SEC先做解密，再做认证校验处理：

软件先解密出最后一个Block的数据以获得Pad_Len，接着更改Len域为原 Len - Len(IVIN) - Len(MAC) - Len(padding) - 1，之后再从头开始做解密+认证。

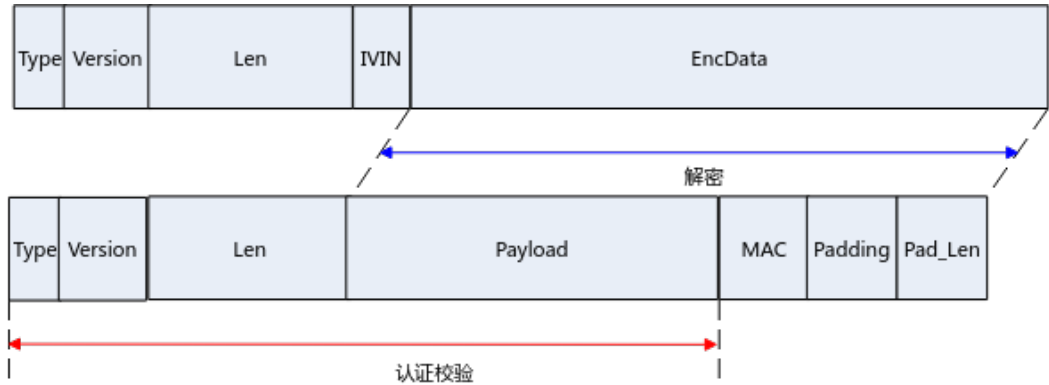
SEC需要对EncData解密出Payload、MAC、Padding、Pad_Len部分。

SEC需要对Sequence Number、Type、Version、Len、Payload部分做认证，算出的MAC值与解密出的MAC值进行比对，比对通过则回写BD中ICV域为1，不通过则回写为2；

SEC在输出时剥除在Len域和Payload数据间的IV；并可据软件配置，进一步剥除Padding、Pad_Len，修改Len域。



图 8-24 OpenSSL/TLS 上行方向 SEC CBC 解密+认证校验处理



TLS_Len_Update域段配置为1；

Cipher_Pad_Len域段填写软件对密文最后一个block解密后的Pad_Len的值；

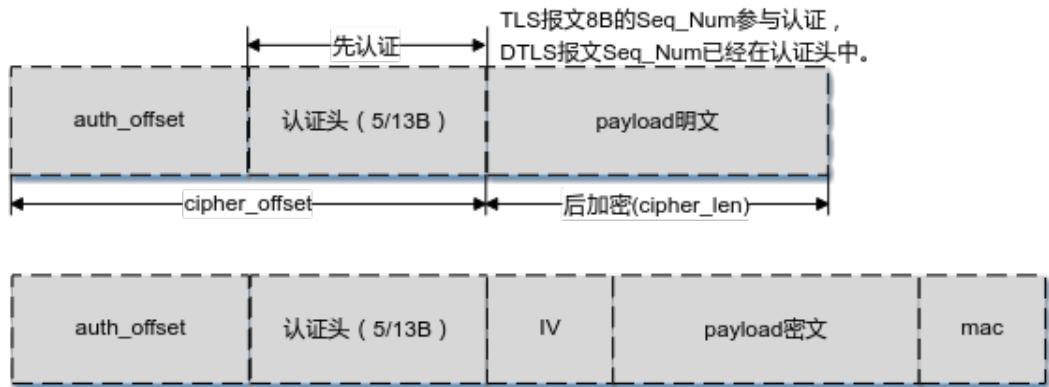
Cipher_Len域段包含payload密文 + mac密文 + pad密文，需根据算法协议block对齐；

Cipher_Offset域段配置不小于报文头长度 + IV长度；

Auth_Offset配置为Cipher_Offset - 报文头长度 - IV长度；

Auth_Len配置为报文头长度 + payload；

图 8-25 OpenSSL/TLS 上行方向 SEC CCM 加密认证



Cipher域段配置为1；

Auth域段配置为0；

DE域段仅配置为1；

SN域段：TLS场景填写Sequence Number，DTLS场景忽略；

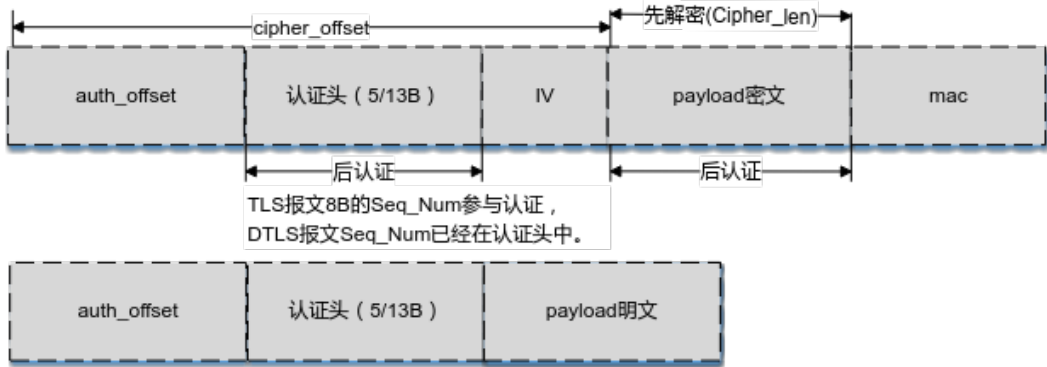
TLS_Len_Update域段配置为1；

Cipher_Len域段配置为payload；源地址空间大小和目的地址空间大小保持一致；

Cipher_Offset域段配置不小于报文头长度；

Auth_Len配置为报文头长度;
Auth_Offset配置为Cipher_Offset - 报文头长度;

图 8-26 OpenSSL/TLS 上行方向 SEC CCM 解密认证



Cipher域段配置为2;
Auth域段配置为0;
SN域段: TLS场景填写Sequence Number, DTLS场景忽略;
TLS_Len_Update域段配置为1;
Cipher_Len域段配置为payload密文 + mac密文; 源地址空间大小和目的地址空间大小保持一致;
Cipher_Offset域段配置不小于报文头长度 + 8B;
Auth_Len配置为报文头长度;
Auth_Offset配置为Cipher_Offset - 报文头长度 - 8B;

图 8-27 OpenSSL/TLS 上行方向 SEC GCM 加密认证

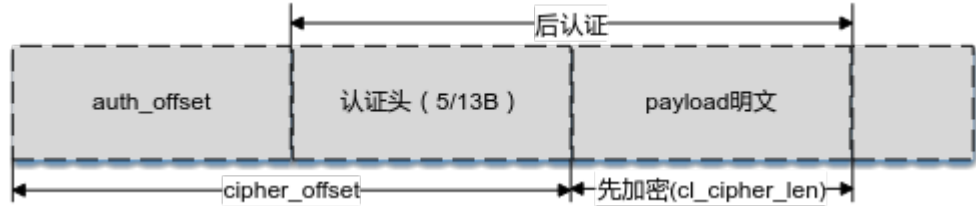
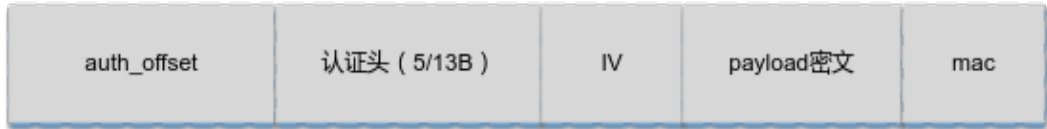


图 8-28 OpenSSL/TLS 上行方向 SEC GCM 加密认证后报文结构



Cipher域段配置为1;

Auth域段配置为0;

SN域段: TLS场景填写Sequence Number, DTLS场景忽略;

TLS_Len_Update域段配置为1;

Cipher_Len域段配置为payload; 源地址空间大小和目的地址空间大小保持一致;

Cipher_Offset域段配置不小于报文头长度;

Auth_Len配置为报文头长度;

Auth_Offset配置为Cipher_Offset - 报文头长度;

图 8-29 OpenSSL/TLS 上行方向 SEC GCM 解密解认证

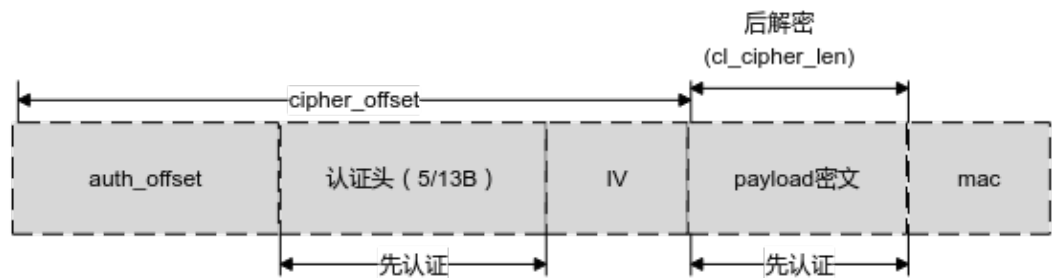
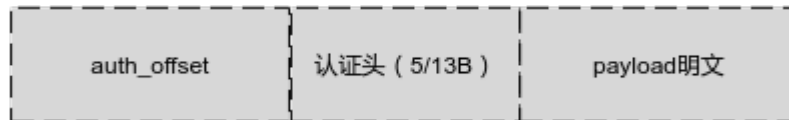


图 8-30 OpenSSL/TLS 上行方向 SEC GCM 解密解认证后报文结构



Cipher域段配置为2;

Auth域段配置为0;

SN域段: TLS场景填写Sequence Number, DTLS场景忽略;

TLS_Len_Update域段配置为1;

Cipher_Len域段配置为payload密文; 源地址空间大小和目的地址空间大小保持一致;

Cipher_Offset域段配置不小于报文头长度 + 8B;

Auth_Len配置为报文头长度;

Auth_Offset配置为Cipher_Offset - 报文头长度 - 8B;

8.5.3 TLS 1.3 机密性和完整性协议处理

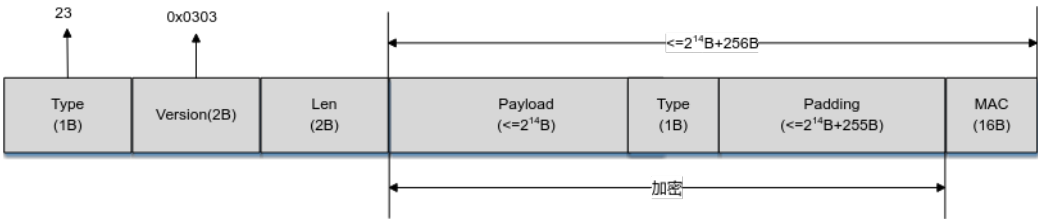
8.5.3.1 SEC 支持的功能介绍

支持TLS1.3协议中规定的AES-GCM、AES-CCM算法做机密性和完整性处理, 其中, AES-GCM算法支持整包调用和流式调用 (又称“流模式”), 而AES-CCM仅支持整包

调用，不支持流式调用；支持TLS 1.3协议加密方向报文payload尾部添加明文Type、Padding的功能；

支持TLS1.3协议解密方向报文payload尾部明文Type的提取和Padding长度的提取；

图 8-31 TLS1.3 SEC 加密方向处理



8.5.3.2 加密方向的软硬件配合

软件封装好TLS1.3报文格式：

1. 报文头部添加Type域段（固定值23）、Version域段（固定值0x0303）、Len域段（含payload、明文Type、Padding、MAC——协议里允许AEAD算法处理后的报文长度增长<=255B，AES-GCM、AES-CCM算法增加的仅是MAC（又称Tag），最长只会到16B）。
2. 预留尾部明文Type域段+Padding+MAC位置。
3. 将明文Type域段+Padding_Len（padding的实际长度）填到SEC BD里，并指示是否填充明文Type域段，是否填充padding（全0）。
4. 将拼接好的IVIN（16B）填写到Cipher_IVIN_Addr地址处（整包调用时，AES-GCM算法还支持软件将拼接好的IVIN（16B）直接填写在BD中，即Tls_1p3_gcm_ivin域）。
5. 软件填写BD：Scence域段配置为4'hA；BD中配置明文Plaintext_Type域段、Padding_Length_1p3（padding的实际长度）、Deal_TLS1p3域段（指示是否需要硬件填Type或Padding）；Cipher_Len域段配置为payload、Type、Padding_Len的和；Auth_len域段配置为5Byte；其他常用域段配置。

SEC硬件完成：

1. 如果SEC BD里指示需要填充Type域段，则将明文Type域段添加到明文之后（加密前填充完成）；如果指示不填充Type域段，则不填。
2. 如果SEC BD里指示需要填充padding，则在Type域段之后添加padding（全0），填充长度遵照Padding_Len（加密前填充完成）；如果SEC BD里指示不需要填充padding，则不填。
3. 支持从Cipher_IVIN_Addr读入软件拼接好的IVIN（16B）（整包调用时，AES-GCM算法还支持从SEC BD中直接获取软件拼接好的IVIN）。

8.5.3.3 解密方向的软件配置

软件填BD：

1. Scence域段配置为4hA。
2. Cipher_Len域段配置为密文长度。
3. Auth_len域段配置为5Byte。

4. 将拼接好的IVIN（16B）填写到Cipher_IVIN_Addr地址处（整包调用时，AES-GCM算法还支持软件将拼接好的IVIN（16B）直接填写在BD中，即Tls_1p3_gcm_ivin域）。
5. 其他域段详见BD描述。

SEC硬件完成：

1. SEC完成报文的解密。
2. SEC回写BD中回写解密后提取的报文尾部Type和Padding长度值；（Tls_1p3_Type_Back：提取出解密后报文尾部最后一个非零值；Padding Length_1p3_Back：从尾部倒着数非零的个数）。

软件更新报文中LEN域段：

1. 软件读取SEC回写BD中的Tls_1p3_Type_Back、Padding Length_1p3_Back域段。
2. 软件根据需要更新报文头部的LEN域段。
3. 软件剥除不需要的TLS1.3报文的相应域段。

8.5.3.4 TLS1.3 GCM 流模式

SEC支持TLS1.3协议GCM的流模式，实现方式同“[流处理模式加解密功能](#)”的步骤。

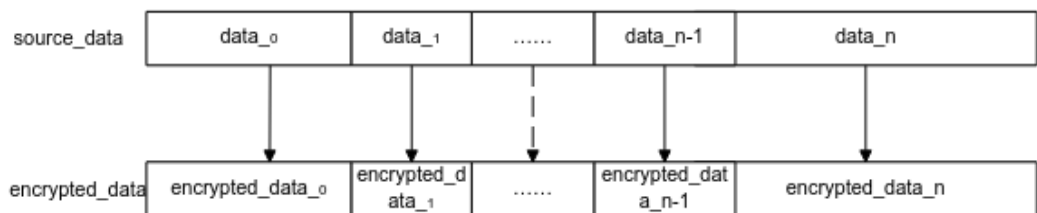
8.5.4 流处理模式加解密功能

流处理模式加密是指业务软件下需要对一个很长的文件进行加密，但是调用加速器的时候分块调用，最后将每块的加密结果组装在一起，组装后的结果和对整个文件一次加密的结果是一致的。如[图8-32](#)所示，source_data为待加密的一个长文件，软件将其分割成n段数据data_0、data_1…data_n-1、data_n分别送给加速器加密，依次得到encrypted_data_0、encrypted_data_1…encrypted_data_n-1、encrypted_data_n等n段加密数据。组成的encrypted_data和对整个source_data的加密结果是一致的，满足协议。

流处理模式的解密方向与加密方向差别仅仅是算法核在处理时为解密计算，其他保持一致。

其中流模式CTR的counter使用上一次BD回写中的值。

图 8-32 IPsec AH 隧道模式下行方向 SEC 处理



GCM算法流模式首中尾BD配置流程，如[表8-4](#)所示。



表 8-4 GCM 算法流模式首中尾 BD 配置流程

-	首BD	中间BD	尾BD
BD配置	Cipher为0， Auth为1， 认证算法配置为AES_GMAC， AI_GEN为0， Auth_Pad为0b1， Akey_Len合法值， Auth_Key_Adr为存放Key的地址， 配置Auth_Src_Offset， Auth_Len大于0， Mac_Adr为输出MAC的地址。	Cipher为1或2， Auth为0， 加解密算法配置为AES_GCM， AI_GEN为1， Auth_Pad为1， Akey_Len与首BD相同， Ckey_Len 域段配置满足 $Akey_Len = 4 + 2 * Ckey_Len$ ， Cipher_Key_Adr同首BD中的Auth_Key_Adr， $Cipher_Len = 16 * N$ （N为正整数）， Cipher_Src_Offset=Auth_Src_Offset， Auth_Len为0。	配置Cipher同中间BD， Auth为0， 加解密算法配置为AES_GCM， AI_GEN为1， Auth_Pad为0， Akey_Len与首BD相同， Ckey_Len配置满足 $Akey_Len = 4 + 2 * Ckey_Len$ ， Cipher_Key_Adr同首BD中的Auth_Key_Adr， Cipher_Src_Offset=Auth_Src_Offset， Auth_Len=0， Cipher_Len大于零， $Long_Auth_Data_Len[23:0] = \text{首BD的Auth_Len}$ ， $Long_Auth_Data_Len[47:24] = \text{所有Cipher_len之和}$ ， 配置Cipher_IVIN_Adr， 配置Auth_IVIN_Adr。
IVIN	无。	Cipher_IVIN_Adr地址存放IVIN{salt，随机数，Counter}， Counter来源于上一个BD的Counter， 配置Auth_IVIN_Adr（建议同上一个BD的Mac_Adr，使用上一个BD输出的MAC值AIVIN）。	尾BD的Auth_IVIN_Adr中存放数据为{MAC(16B),null(16B)},{salt,随机数，Counter}}从左到右由高到低， Cipher_IVIN_addr中存放{salt，随机数，Counter}。
KEY	保持一致。	保持一致。	保持一致。
注意事项	-	-	建议Auth_IVIN_Adr与Cipher_IVIN_Adr不在同一个128B地址空间。
ESP Padd ing	-	-	Stream_Protocol配置为1。



-	首BD	中间BD	尾BD
TLS1.3 Padding	-	-	根据需要Stream_Protocol配置为2、3、4。

须知

GCM流模式支持cipher_len最大到16M~512B。

CTR算法流模式首中尾BD配置流程，如表8-5所示。

表 8-5 CTR 算法流模式首中尾 BD 配置流程

-	首BD	中间BD	尾BD
BD配置	Cipher为1或2， Auth为0， 加解密算法配置为AES_CTR， AI_GEN为0， Auth_Pad为0， Scene配置为0x7流式数据处理 Cipher_Key_Addr为存放Key的地址 Cipher_Len = 16*N （N为正整数）， Counter域段填 0x1。	Cipher为1或2， Auth为0， 加解密算法配置为AES_CTR， AI_GEN为0， Auth_Pad为0， Scene配置为0x7流式数据处理 Cipher_Key_Addr同首BD中的 Cipher_Key_Addr， Cipher_Len = 16*N （N为正整数）， Counter域段值来源于上个BD回写的Counter值。	Cipher为1或2， Auth为0， 加解密算法配置为AES_CTR， AI_GEN为0， Auth_Pad为0， Scene配置为0x7流式数据处理 Cipher_Key_Addr同首BD中的 Cipher_Key_Addr， Cipher_Len（无整数倍限制）， Counter域段值来源于上个BD回写的Counter值。
IVIN	Cipher_IVIN_Addr地址存放IVIN{salt，随机数，Counter}。	Cipher_IVIN_Addr地址存放IVIN{salt，随机数，Counter}， Counter来源于上一个BD的Counter。	Cipher_IVIN_Addr地址存放IVIN{salt，随机数，Counter}， Counter来源于上一个BD的Counter。
KEY	保持一致。	保持一致。	保持一致。
注意事项	-	-	-
ESP Padding	-	-	-



-	首BD	中间BD	尾BD
TLS1.3 Padding	-	-	-

8.5.5 长数据 Hash 功能

SEC支持MD5、SHA1、SHA224、SHA256、SHA-384、SHA-512、SHA-512/256、SHA-512/224、SM3以及HMAC_MD5、HMAC_SHA1、HMAC_SHA224、HMAC_SHA256、HMAC_SHA-384、HMAC_SHA-512、HMAC_SHA-512/256、HMAC_SHA-512/224、HMAC_SM3算法下的长数据Hash。这些算法做长数据Hash时最大可运算 $(2E - 1)$ Byte (即 $2^{64} - 8\text{bit}$) 的数据。

一段长报文，需要被分割为1个首Hash BD，若干个中间Hash BD，1个尾Hash BD配置给SEC。每个分片最大长度为16MB-513B，除最后一个分片外，其余所有分片长度都必须为64B整数倍。前一个BD输出的MAC，需要做为后一个BD的IVIN，因此这些分割出的BD之间需要原子保序（即前一个BD处理完毕后，软件才能下发后一个BD）；软件保证这些BD之间被SEC处理的顺序。

在配置首Hash部分时要求：BD配置AI_GEN域为0b0，AUTH_PAD域为0b1，MD5、SHA1、SHA224、SHA256以及HMAC_MD5、HMAC_SHA1、HMAC_SHA224、HMAC_SHA256，这些算法下要求Auth_Length为64Byte的整数倍，SHA-384、SHA-512、SHA-512/256、SHA-512/224以及HMAC_SHA-384、HMAC_SHA-512、HMAC_SHA-512/256、HMAC_SHA-512/224，这些算法下要求Auth_Length为128Byte的整数倍；

在做中间部分Hash时要求BD配置AI_GEN域为0b1，AUTH_PAD域为0b1，Auth_Length要求同首BD时。

在配置尾Hash部分时要求，BD配置AI_GEN域为0b1，AUTH_PAD域为0b0，Auth_Length长度允许字节对齐，同时需要配置Long_Auth_Data_Len = 整个长数据Hash运算的报文总长度，总长度以bit为单位（但长度需要保证是字节的整数倍）；

说明

- 长数据Hash功能主要用于：
 - 待处理数据长度（Auth_Source_Offset+Auth_Len）超过SEC BD支持的单次处理数据最大长度(16M-513)B时的Hash计算。
 - 待计算数据不能一次性准备好时的Hash计算。
- 把一个长的Hash计算分成多个BD计算，当前BD依赖于上一个BD的计算结果；因此属于同一个长数据Hash计算的多个BD，需要严格按照先后顺序进行处理。
- 只能在尾BD截取MAC，首BD和中间BD必须配置输出全长度的MAC值，以便做为后续BD的IVIN：MD5、HMAC_MD5的全MAC长度为128bit；SHA1、HMAC_SHA1的全MAC长度为160bit；SHA224、SHA256、HMAC_SHA224、HMAC_SHA256的全MAC长度为256bit；SHA-384、SHA-512、SHA-512/256、SHA-512/224、HMAC_SHA-384、HMAC_SHA-512、HMAC_SHA-512/256、HMAC_SHA-512/224的全MAC长度为512bit。



须知

长数据Hash功能有如下限制：

1. 计算长数据Hash运算的BD，不支持同时处理加解密（也不支持同时处理hardcopy）。
2. 长数据Hash任务计算的原子性，需软件保证。

8.5.6 多模隔离

- SEC加速器支持5个VF（Virtual Function）。
- 支持SEC加速器通过页表进行各模式的权限检查和隔离。
 - SEC加速器访问内存和其他加速器通过SMMU页表隔离。
 - SEC在访问内存空间和其他加速器（POE/BMU）时，需要将streamid随地址一起发送到SMMU，SMMU基于streamid进行权限检查。
 - SEC加速器PF/VF的streamid在PF中配置。
- SEC加速器通过PF-VF的方式进行功能及寄存器编程接口的划分。
 - 物理功能（Physical Function，PF）用于管理所有VF的公共功能以及各个VF之间的资源分配。虚拟功能（Virtual Function，VF）是能够独立完成业务处理的功能（如与队列相关的配置寄存器，运行过程需要频繁操作的寄存器等等）。模块支持1个PF和多个VF，多个VF共享模块处理能力。
 - PF和VF的寄存器支持按照页表粒度（64K）隔离。
 - 支持通过PF相关接口可以完成模块的所有功能，且性能可以达到模块最大。
 - 支持通过配置PF寄存器可以完成VF所有资源的初始化、回收和动态/静态调整。

说明

VF侧读PF寄存器，读清类的寄存器可以读清。

- SEC加速器支持通过VFID对PF/VF所有的任务、总线操作、内部处理数据流等进行标识。
 - SEC支持内部不同模式的任务输入队列、任务输出队列基于VFID隔离。
 - SEC支持输入任务队列处理在不同模式之间按照WRR调度；支持输出任务队列在不同模式之间按照RR调度的方式向POE添加。
 - SEC支持不同模式任务处理独立使能（某一模式任务处理被关闭后，该模式的输入任务不再参与调度）。
- 资源回收。
 - SEC的任务输入队列、任务输出队列支持独立使能（任务输入队列被关闭后，不再向POE获取任务；任务输出队列被关闭后，不再向POE添加任务）、独立的软件清空和回收（任务输入队列被关闭且该VF的任务都已ADD POE结束后，有状态信号置起，软件自己选择将以ADD到POE中的任务丢弃还是回收）。
- 中断上报。
 - PF中断统一上报，各VF的中断独立上报，一个模式一根异常中断线。
 - 不支持VF中断注入。
 - 多个中断一旦触发，会同时上报PF和VF，实际应用时建议只选择一方上报。



- 参考《Hi309a 寄存器手册》使用SEC_PF_REG页签的SEC_PF_ABNORMAL_ENABLE寄存器和SEC_POE_VF_REG页签的SEC_POE_VF_ABNORMAL_ENABLE寄存器来选择是否使能对应的中断。
- vf_task_time_out_rint: 当出现该中断时, 往往意味着整设备挂死, 需要模块复位才能解决, 所以对该中断建议使能PF页签的相关寄存器, 而VF页签的寄存器可使能或不使能(如果VF页签使能上报, 建议VF软件做停流等处理)。
- vf_bmu_get_fail_rint: SEC模块不与BMU模块交互, 建议VF页签的相关寄存器不使能。
- vf_auth_check_fail_rint、vf_bd_err_rint: 对于认证校验错误和BD解析错误, 期望软件对每个BD的回写状态进行检查并相应处理, 不能依赖上报中断; 建议只使能PF页签相关寄存器。

8.5.7 SEC 大小端翻转功能

网络报文是以大端方式存储在DDR中, 即一段报文的最高字节(最重要字节)存在最低地址, 而SEC内部的大多数算法Core都是以小端方式处理。所以SEC内部提供了灵活的大小端翻转方式, 确保加解密的正确运行。

以一段16Byte的报文为例子, 报文为0x00112233_44556677_8899AABB_CCDDEEFF, 当对这段报文进行DES加密处理时候, 因为DES算法的Block长度是8Byte, 即每次能处理的数据是64bit, 实际对这段报文的处理流程应该是先对0x00112233_44556677进行加密, 再对0x8899AABB_CCDDEEFF, 这是期待的正确的加密顺序。

假如这段报文以大端方式存储在DDR中, 那么它的存储方式如表8-6所示, 地址从0x0000开始依次由下至上递增, 每个地址存储的是1个字节数据。

表 8-6 报文在 DDR 中大端存储方式

地址	数据
0x000f	ff
0x000e	ee
0x000d	dd
0x000c	cc
0x000b	bb
0x000a	aa
0x0009	99
0x0008	88
0x0007	77
0x0006	66
0x0005	55
0x0004	44
0x0003	33



地址	数据
0x0002	22
0x0001	11
0x0000	00

当SEC从DDR中将该段数据读到内部FIFO后，FIFO的位宽按64bit存储，则存储在FIFO的第1个地址的数据是0x77665544_33221100，第2个地址的数据是0xFFEEDDCC_BBAA9988，对于DES算法，每次从FIFO的读取1个64bit数据，就启动一次运算，这样从FIFO中读到的第1个数据就是0x77665544_33221100，但实际送到DES CORE里去加密的数据应该是0x00112233_44556677，因此SEC内部会对这个数据进行翻转，对于DES算法来说，翻转就是基于64bit（DES的Block长度）进行BE-8的翻转。

假设当对这段报文进行AES加密处理时候，因为AES算法的Block长度是16Byte，即每次能处理的数据是128bit，实际对这段报文的处理流程应该是先对0x00112233_44556677_8899AABB_CCDDEEFF进行加密，这是期待的正确的加密顺序。

AES算法处理时候，每次从FIFO中读2个数据启动一次运算，从FIFO中读到的128bit数据为0xFFEEDDCC_BBAA9988_77665544_33221100，送到AES CORE的数据应该是0x00112233_44556677_8899AABB_CCDDEEFF，因此SEC内部会对这个数据进行翻转，对于AES算法来说，翻转就是基于128比特（AES的Block长度）进行BE-8的翻转。

SEC内部在Hash算法以及一些需要填充的算法中，通常在最后1个Block中，仅仅需要对于实际数据进行翻转，而不需要对于填充部分数据进行翻转。

访问SEC寄存器只支持小端模式；SEC通过AXI Master接口访问报文相关数据（含报文、KEY、IVIN、MAC等）时只支持大端模式；访问BD解析描述符支持大小端可配，配置寄存器为PF SEC_CONTROL（偏移地址0x200）中的bit[3:2]：支持32bit大端（在32bit内进行字节序翻转，32bit之间的顺序保持不变）和64bit大端（在64bit内进行字节序翻转，64bit之间的顺序保持不变）两种模式寄存器可配置，大小端配置默认为大端。

8.5.8 IVIN 的获取

SEC在IPSEC、OpenSSL/TLS场景下加解密两个方向都支持直接使用软件拼接好的IVIN：软件将拼接好的IVIN放在Cipher_IVIN_Addr处（特别地，AES-GMAC算法的IVIN是放在Auth_IVIN_Addr处），SEC从该地址读取完整的IVIN使用。

SEC在IPSEC、OpenSSL/TLS场景下加解密方向也都支持自己使用相关元素组装出加解密IVIN（特别地，AES_CCM、AES_GCM算法也会生成认证IVIN，而AES_GMAC算法只会生成认证IVIN），之后使用该IVIN进行运算：

拼接IVIN，加密方向需要用到从TRNG申请到的随机数，还可能用到软件随SEC BD配置下来的其他IVIN组成元素；并且计算完成后还会写出组成IVIN的随机数：使用Cipher_IVIN_Addr地址（特别地，AES-GMAC算法是Auth_IVIN_Addr）。

拼接IVIN，解密方向需要用到从Cipher_IVIN_Addr处（特别地，AES-GMAC算法是Auth_IVIN_Addr）读入的随机数，还可能用到软件随SEC BD配置下来的其他IVIN组成元素。



表 8-7 IPSEC、OpenSSL/TLS 场景下 IVIN 组成方式

Alg	随机数	SEC BD的 salt域	其他	加解密IVIN组成方式	认证IVIN组成方式
DES_ECB 3DES_ECB AES_ECB	-	-	-	-	-
DES_CBC 3DES_CBC	前8B 随机数	-	-	{随机数}	-
AES_CBC	前16B 随机数	-	-	{随机数}	-
AES_OFB	前16B 随机数	-	-	{随机数}	-
SM4_CBC	前16B 随机数	-	-	{随机数}	-
AES_CTR	前8B 随机数	4B（其实是nonce）	Counter（0x0000_0001）	{Nonce，随机数，Counter}	-
AES_CCM	前8B 随机数	IPSEC：3 B（salt[23:0]） TLS：4B（salt[31:0]）	IPSEC：Counter（0x0000_0001）； TLS：Counter（0x0000_0001）； Flag（1Byte，需自己计算，加解密和认证时定义不同），	IPSEC：{ Flag（1B），salt（3B），随机数(8B)，Counter(4B) }； TLS：{ Flag（1B），salt（4B），随机数，Counter(3B) }；	IPSEC：{ Flag(1B)，salt(3B)，随机数(8B)，报文长度(4B) }； TLS：{ Flag(1B)，salt(3B)，随机数(8B)，报文长度(4B) }；
AES_GCM	前8B 随机数	4B	Counter（0x0000_0001）	软：{salt，随机数}	软：{salt，随机数}
				硬：{salt，随机数，Counter }	硬：{salt，随机数，Counter }
AES_GMAC	前8B 随机数	4B	Counter（0x0000_0001）	-	软：{salt，随机数}



Alg	随机数	SEC BD的 salt域	其他	加解密IVIN组成方式	认证IVIN组成方式
					硬：{salt, 随机数, Counter }

8.5.9 UDP checksum 域计算

8.5.9.1 补码算法

UDP Checksum使用one’s complement（补码）运算。运算位宽为16bit。其中补码加法运算规则如图8-33所示。对于没有进位的运算，运算规则与普通加法相同，对于有进位的运算，还需要将进位加到结果上。

图 8-33 补码加法运算规则

0001 0110	22	0001 0110	22
+ 0000 0011	3	+ 1111 1111	-0
=====	=====	=====	=====
0001 1001	25	1 0001 0101	21
		+ 0000 0001	1
		=====	=====
		0001 0110	22

8.5.9.2 UDP Header 介绍

图 8-34 UDP 头结构

UDP Header																																	
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

- Source port：源端口。
- Destination port：目的端口。
- Length：UDP头长+UDP数据长度，以Byte为单位。
- Checksum：如果该域不使用，checksum域填0。如果checksum域计算出来是全0，应该以全1发送。计算checksum时，初始值应该填0，计算后回填此域。



8.5.9.3 IPv6 Header 介绍

表 8-8 IPv6 Header 结构

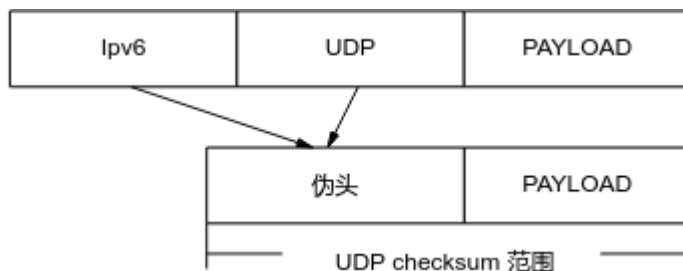
3	3	2	2		2	2	2	2	2	2	2	2	2	1	1	1	1		1	1	1	1	1	1	9	8	7		6	5	4	3	2	1	0						
1	0	9	8		7	6	5	4	3	2	1	0		9	8	7	6		5	4	3	2	1	0																	
Version n				Traffic class										Flow label																											
Payload length																		Next header										Hop limit													
Source address																																									
Source address																																									
Source address																																									
Source address																																									
Destination address																																									
Destination address																																									
Destination address																																									
Destination address																																									

- Version: 4bit, IP版本号, 数值为6。
- Traffic class: 8bit, 业务类型, 用来区分IP分组的类型和优先级。
- Flow label: 20bit流标签, 发送方对于需要路由器特殊处理的分组打上顺序的流标签。
- Payload length: 16bit无符号整数。IP包净荷长度, 不包括IP头部。
- Next header: 8bit选择器。确定接在IPv6头部后的头部类型, 相当于IPv4中的协议域。
- Hop limit: 8bit无符号整数。每经过一个路由器减一, 如果跳数减到0, 分组会被丢弃。
- Source address: 128bit源地址。
- Destination address: 128bit接收端地址。

8.5.9.4 IPv6 + UDP 应用

从伪头的第一个半word开始, 计算到UDP包结束。如图8-35所示。由于计算的单位是半word, 对于数据总长度有奇数偶数之分。当总长度为偶数时, 计算公式为[A,B] + [C,D] + ... + [Y,Z]。对于奇数时, 计算公式为[A,B] + [C,D] + ...+[Z,0]。其中A表示一个Byte。

图 8-35 UDP checksum 数据格式



其中伪头的格式如图8-36所示。

图 8-36 IPv6 伪头格式

IPv6 Pseudo Header Format																																	
Offsets	Octet	0						1						2						3													
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															

特别提醒一下，伪头中的next header域应该使用上层协议头的值，此处即UDP头的协议值；在IPv6头和UDP头之间存在扩展头时要特别注意此点。

须知

按照协议规定，在IPv6头和UDP头之间存在扩展头时，图8-36伪头中的Next Header域仍应该使用上层协议头的值，即UDP头的协议值。

8.5.9.5 Check Sum 计算

Check Sum计算涉及SEC BD中的4个域：CS、CSC、CS_IP_Header_Offset、CS_UDP_Header_Offset，各域在SEC BD中的位置及具体含义见**表8-9**中相关描述。

Check Sum计算对象为原始报文，所以隧道模式下使用的是内部IP Header和payload。

在加密方向（CS=1），先计算Check Sum，后进行加密、认证等处理；在解密方向（CS=2），先进行解密、解认证等处理，后计算Check Sum。

计算Check Sum时，先通过Data_Base_Addr（直接地址或者PPE链、SGL链里所存数据的首地址）+ CS IP Header Offset得到待计算IP头的起始位置，从IP头中提取



8-36中所示“棕色”的域段顺次进行Check Sum计算（注意，只对这几个域做计算，不对全部IP头做计算）。

再通过Data_Base_Addr + CS_UDP_Header_Offset得到待计算UDP头的起始位置，获取UDP头的Length域做为待计算的UDP报文长度（UDP头长+UDP数据长度），之后使用IP头中相关域的Check Sum结果继续对全部UDP报文进行Check Sum计算，从而得到最终的Check Sum结果。

如果是加密方向（CS=1），将算出的Check Sum写入UDP头的Checksum域；如果是解密方向（CS=2），将算出的Check Sum与UDP头的Checksum域进行比较，比较结果通过SEC BD的CSC域回写。

值得一提的是，在解密方向（CS=2），向SEC下发任务的软件在计算CS_UDP_Header_Offset时，对IP头需要始终按照无扩展头来看待。而在SEC处理完成后重新交给软件时，软件需要查看SEC回写的CSC域，如果为2'b01 check成功则软件不用做后处理，如果为0b10 check失败，则软件需要自己解析IP头看其是否有扩展头：若有扩展头，则SEC对Check Sum的校验结果不可信，软件需要自己重新做Check Sum的校验；若没有扩展头，则SEC对Check Sum的校验结果可信，即报文已被改动，或者存在Check Sum相关配置错误。

8.5.9.6 Check Sum 注意事项

软件配置时需要满足：

$CS_UDP_Header_Offset - CS_IP_Header_Offset \geq 40$;

UDP头中的Length域 ≥ 8 ;

UDP头中的Length域 $\leq MAX_Length$;

UDP头的Length域 + $CS_UDP_Header_Offset \leq OFFSET + MAX_Length$;

其中，

C_S=1时：

Auth == 1或者2时， $MAX_Length = Auth_Length$, $OFFSET = Auth_Source_Offset$;

Auth=0时，且Cipher=0或1或3时 $MAX_Length = Cipher_Length$, $OFFSET = Cipher_Source_Offset$;

C_S=2时：

Auth == 1或者2时， $MAX_Length = Auth_Length$, $OFFSET = Auth_Source_Offset$;

Cipher == 2时， $MAX_Length = Cipher_Length$, $OFFSET = Cipher_Source_Offset$;

Cipher == 3 或者Cipher == 0且Auth==0时， $MAX_Length = Chain_Length$, $OFFSET = 0$ 。

此外，在C_S=2时，若Auth == 1或者2且Cipher == 2时，还需满足 $Auth_Length + Auth_Source_Offset = Cipher_Length + Cipher_Source_Offset$;

8.5.10 密钥管理

8.5.10.1 密钥类型

SEC上电复位后通过接口向KM申请Wrapping KEY、STORAGE_KEY两种类型的KEY，每种KEY的位宽都是256bit。



Wrapping KEY: 用来解密DDR中存放的KEY密文的KEY;

STORAGE_KEY: 安全任务下, 用来解密DDR中存放的KEY密文的KEY。

8.5.10.2 功能特性

安全软件控制SEC对三种KEY的进行加载, SEC提供对某一个key单独加载和对所有key全部加载的功能; 详细参考SEC_BD_FIFO页签SEC_BDFIFO_REG中KM_REQ_DONE、KM_AUTO_REQ_START寄存器的使用。

- SEC具有请求超时中断上报的功能, 当SEC向KM发起请求后长时间未得到KM响应时, SEC会向软件上报超时中断。
- SEC具有维护三种KEY的使用次数和CRC检测正确性, 检测到CRC错误会上报中断, 且回写BD中带上相应错误类型。
- SEC对认证和校验的权限管理比较简单, 当KEY有效时, 即可做认证和校验业务。
- SEC具有检测Wrapping KEY、STORAGE_KEY并错误上报的功能, SEC使用Wrapping KEY解密密文KEY时, 如果认证失败则会上报中断。
- SEC具有检测越权使用KEY的BD (包括使用没有获取权限的KEY, 还未加载就使用这些KEY), 并在回写BD中标识这种错误类型。
- SEC具有解密KEY总线性能优化的特性, SEC使用Wrapping KEY、STORAGE_KEY解密DDR中的密文KEY时, 通过KEY地址和VF判断是否需要重新解密密文KEY。

SEC与系统约定当使用Wrapping KEY、STORAGE_KEY时, KEY地址中存放IV、A、密文KEY和TAG数据。

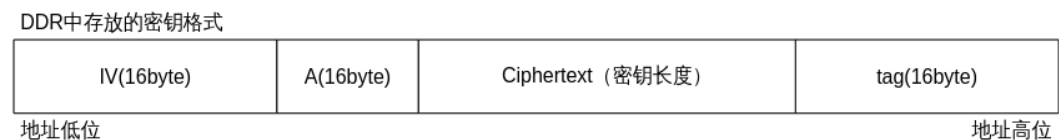
SEC使用STORAGE_KEY具有严格的权限控制, 使用必须同时满足下面三个条件: KM返回这个KEY时指示为有效、BDFIFO下发的安全任务、BD中选择了这个KEY; 当KM指示这个KEY无效, 但BDFIFO下发的这个BD (有效BD) 中选择了这个KEY, BD回写BD_ERR, 且SEC上报中断。

Wrapping KEY可以在安全或非安全任务中被使用。

8.5.10.3 DDR 中密文 KEY 的数据格式

基于NIST SP800-38D标准的AES-GCM算法用来对密钥进行加密, 可以同时密钥进行机密性和完整性保护。为了对wrapped_key进行完整性保护, 基于AES-GCM生成的wrapped_key方式, 都需要提供IV值 (16bytes, 高12bytes有效, 低4bytes填0) 和additional authenticated data A (16Byte)。KM对明文密钥进行加密之后, 会将IV, A, wrapped_key以及16bytes Tag回写到结果地址。图中的密钥长度为密钥本身长度, 不包括IV, A以及tag长度。

图 8-37 DDR 中密文 KEY 的数据格式



8.5.11 一级 UNWRAP

支持用明文的wrapping_key对业务密钥的密文解密, 只需要做一次AES-GCM-256解密操作得到业务密钥明文。该过程简称“一级UNWRAP”。

8.5.11.1 一级 UNWRAP 操作

图 8-38 一级 UNWRAP 数据格式与操作流程图

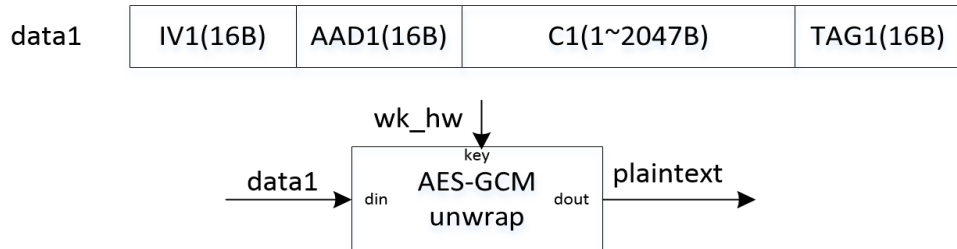


图8-38中data1表示业务密钥的密文，wk_hw表示硬件wrapping_key，plaintext表示业务密钥的明文。

密文数据：软件需严格按照上图8-38data1的顺序准备输入数据，IV固定16bytes。实际参与计算的IV 16 byte构成为Nonce+pad=12+4=16bytes，或者说是salt+IV+pad=4+8+4=16 bytes，这两种说法表述一致，均只采用了输入IV的前12byte），AAD固定16bytes,业务密钥密文长度支持1~2047bytes（SEC只需要支持1~1024bytes），TAG固定16bytes。BD需指定业务密钥的长度和存储地址。

密钥：软件在BD中指定做UNWRAP的密钥（图中的wk_hw），可以来自KM传递给加速器的WRAPPING_KEY。密钥长度固定256bits。BD中需指定选择哪种密钥来密。

8.5.12 低功耗

1. SEC支持对SAA时钟的关断。
2. SEC支持各算法核在没有处理数据时时钟被硬件自动关断，此功能有独立使能位。

8.5.13 透传包处理

SEC支持软件生成的透传包（BD1里是Cipher=0，Auth=0，BD2/BD3里是Cipher=0，Auth=0且C_S=0）的处理：只读写BD，不读写任何其他数据（含IVIN、KEY、MAC、报文等），并经队列保序后，将任务输出。

8.5.14 BD IVLD

当BD中的BD_IVLD域配置为1，表征当前BD不进行处理，SEC仅读取和回写BD，但是不会对报文内容进行任何处理。

8.6 编程接口



须知

- Hi309a SEC采用新BD结构（请参见[Buffer指针结构](#)），POE和BD_FIFO两种工作方式配置方式不同，具体如下：(1) POE任务下发，需要在输入包任务描述符（见1.6.2章节）中的bd_type_sel域段选择新BD。(2) BD_FIFO任务下发，需要在《Hi309a 寄存器手册》，SEC_BDFIFO_REG页签，配置寄存器SEC_BDF_CFG_BD_TYPE（偏移0x008），为0b1（启用BD TYPE 3）。
- Hi309a SEC采用新的地址回写方式，相关寄存器默认值已配好。

POE/BD_FIFO配置为新方式时：

- 当新BD结构（见1.6.1章节）DE域段为0b0时：
 - POE/BDFIFO BD回写地址为cfg_addr（原址回写）。
- 当新BD结构（见1.6.1章节）DE域段为0b1时，写到新地址：
 - POE/BDFIFO BD回写地址为dst_addr（见1.6.1章节）。
 - POE 报文回写到dst_addr + vf_skip_length_cfg（见SEC寄存器描述）。
 - BDFIFO报文回写到dst_addr + sec_bdf_cfg_skip_length（见SEC寄存器描述）。

注意：不支持PPE链式报文回写到新地址。

输出包任务描述符（请见[维测信息](#)）中NB域段会指示是否有搬移到新地址。

8.6.1 SEC BD（SQE）

表 8-9 SQE 数据结构（除存储重删及上下盘加解密以外的场景，如下为小端格式描述）

-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作																															-
-																																-
-	3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
W o r d 0	PrevAddr[31:0]/TimeStamp_NanoSecond[31:0]																															P C E 下 发 的 B D 解 析
W o r d 1	PrevAddr_Reserved/ TimeStamp_Second[31:16]															PrevAddr[47:32]/ TimeStamp_Second[15:0]																
W o r d 2	NextAddr[31:0]																															



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作															-
-																-
Word 3	WNextAddr_Reserved							NextAddr[47:32]								
Word 4	WRefltselAddr[31:0]															
Word 5	WRefltselAddr_Reserved							RefltselAddr[47:32]								
Word 6	WDataOffset							DataLength								
Word 7	WChainOffset							T_Len=6b011000			T_TagValue=6b000010			Dsc_Le n=4b0010		
Word 8	WLocation	Vqi_Rsv	Vqi[9:0]				Err_Vqi_Rsv	Err_Vqi[9:0]				Reserved				
Word 9	WReserved	Order_Rsv	Order_Type		Order_TagValue[23:0]											
Word 10	WReserved				MAC_Addr_Type	Dst_Addr_Type	Src_Addr_Type	DE	Scene		BID_VLD	BD_Type=3				



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作												-
-													-
Word 11	Reserved	SEQL - IV - Addr - End	Reserved	Nonce_Len	C_Algo	C_Mode	CKey_Len	C_Width	C_ICV_Len	CI_GEN	Cipher	通用配置	
Word 12	WCipher_Key_Addr[31:0]												
Word 13	WCipher_Key_Addr[63::32]												
Word 14	Reserved	Update_Key	KEY_SE L	A_Algo		AKey_Len		MAC_Len	AI_GEN	Auth			
Word 15	WSalt[7:0]			Salt[15:8]		Salt[23:16]		Salt[31:24]					
Word 16	WCipher_Src_Offset						Auth_Src_Offset						



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
Word 17	W Auth_Key_Offset	Auth_Len	IPSEC
Word 18	W Auth_IVIN_Offset	Cipher_Len	
Word 19	W Data_Dst_Addr[31:0]		
Word 20	W Data_Dst_Addr[63:32]		
Word 21	W MAC_Addr[31:0]		
Word 22	W MAC_Addr[63:32]		
Word 23	W Cipher_IVIN_Addr[31:0]		



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作								-
-									-
Word 24	Cipher_IVIN_Addr[63:32]								
Word 25	Padding Length	Next Header	Reserved	IP_Type	Mode	Protocol_Type	Deal_ESP_AH	C_S	
Word 26	Reserved			IV_Offset[15:0]					
Word 27	CS_UDP_Header_Offset			CS_IP_Header_Offset					
Word 23	Cipher_IVIN_Addr[31:0]								PBKDF2
Word 24	Cipher_IVIN_Addr[63:32]								
Word 25	Reserved	PBKDF2_Salt_Len[23:0]							



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
Word 26	Reserved	C_Num	
Word 27	DK_Len		Pass_Word_Len
Word 23	Cipher_IVIN_Addr[31:0]		
Word 24	Cipher_IVIN_Addr[63:32]		
Word 25	Long_Auth_Data_Len[31:0]		
Word 26	Long_Auth_Data_Len[63:32]		



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作						-
-							-
Word 27	Padding Length_1p3[15:8]/Padding Length	Padding Length_1p3[7:0]/Next Header	Plaintext_Type	Reserved	Stream_Protocol	Auth_Pad (Stream_Pha se)	
Word 23	Cipher_IVIN_Addr[31:0]						TLS / SSL / DTLS
Word 24	Cipher_IVIN_Addr[63:32]						
Word 25	WSN [31:0]						
Word 26	WSN [63:32]						



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作					-
-						-
Word 27	Reserved	TLS_Len_Fld	Cipher_Pad_Data_Type	Cipher_Pad_Len	Cipher_Pad_Type	
Word 23	Cipher_IVIN_Addr[31:0]/Tls_1p3_gcm_ivin[95:64]					TLS 1.3
Word 24	Cipher_IVIN_Addr[63:32]/Tls_1p3_gcm_ivin[127:96]					
Word 25	Auth_IVIN_Addr[31:0]/Tls_1p3_gcm_ivin[31:0]					
Word 26	Auth_IVIN_Addr[63:32]/Tls_1p3_gcm_ivin[63:32]					
Word 27	Padding Length_1p3		Plaintext_Type	reserved	Deal_TLS_1p3	



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作						-
-							-
Word 23	W Cipher_IVIN_Addr[31:0]/LBA[31:0]						存储
Word 24	W Cipher_IVIN_Addr[63:32]/LBA[63:32]						
Word 25	W Gen_APP_Val	Gen_VER_Val	Gen_APP_Ctrl	Gen_VER_Ctrl	Gen_GRD_Ctrl	Gen_Page_Pad_Ctrl	
Word 26	W Private_Info						
Word 27	W Block_Size		Chk_RE F_Ctrl	Chk_GRD_Ctrl	reserved	Page_Pad_Type	Gen_REEF_Ctrl
Word 23	W Cipher_IVIN_Addr[31:0]						其他
Word 24	W Cipher_IVIN_Addr[63:32]						



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作						-
-							-
W o r d 2 5	Reserved						
W o r d 2 6	Reserved						
W o r d 2 7	Reserved						
回 写 W o r d 2 8	Warning_Type	Error_Type	Reserved	Flag	CSC	ICV	D o n e
回 写 W o r d 2 9	MAC_I[7:0]	MAC_I[15:8]	MAC_I[23:16]		MAC_I[31:24]		
回 写 W o r d 3 0	Reserved		CheckSum_I				



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作			-
-				-
回 写 W o r d 3 0	Padding Length_1p3_Back	Reserved	Tls_1p3_Type_Ba ck	
回 写 W o r d 3 1	Counter			
-	POE下发的BD			-
-	Dsc_ Len	指示变长报文描述符长度，单位512bit，(0:256bit, 1:512bit)；		-
-	T_ Ta gVal ue	数据结构按照TLV方式定义，类型标识；		-
-	T_ Le n	数据结构按照TLV方式定义，结构长度，单位32bit；		-
-	Prev Addr	对于链表方式存储的报文，该域表示上一段链表节点结构所在地址（按Cache-Line对齐）；		-
-		注意：该部分为软件或其他加速器预留，SEC只实现单向链表。		-
-	Next Addr	对于链表方式存储的报文，该域表示下一段链表节点结构所在地址（按Cache-Line对齐）；		-
-	Refl selfA ddr	该分片报文数据BUFF块基地址,该地址+DataOffset是报文数据起始位置，0表示首节点无数据，只允许首节点不带报文；		-
-	Data Leng th	表示该Buffer报文数据的长度，单位：Byte		-
-	Data Offs et	报文数据首地址相对于DataBuffBaseAddr的偏移，单位：Byte。		-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	ChainOffset	该片报文数据相对于整报文的偏移，单位：Byte。（该域为软件预留部分）	-
-	TimestampNanoSec[31:0]	报文时间戳，纳秒级时间；来自PPE接收描述符首片节点数据结构Word12，SEC透传；	-
-	TimestampSec[31:0]	报文时间戳，秒级时间；来自PPE接收描述符首片节点数据结构Word13，SEC透传；	-
-	ErrVqi[9:0]	Vqi替代原先的(Group,Qid)，POE使用Vqi作为VF的Vqi路由表的相对偏移，访问路由表获取虚拟Qid和虚拟GROUP，并完成虚拟Group到物理Group的转换（SEC用来构造ADD_PACKET数据结构）；遇到错误的任务在ADD POE时使用的Vqi域。（POE下发的BD中需要添加该域段）	-
-	ErrVqiRsv	保留用作规格扩展（SEC用来构造ADD_PACKET数据结构）；遇到错误的任务在ADD POE时使用的Vqi的保留域。	-
-	Vqi[9:0]	Vqi替代原先的(Group,Qid)，POE使用Vqi作为VF的Vqi路由表的相对偏移，访问路由表获取虚拟Qid和虚拟GROUP，并完成虚拟Group到物理Group的转换（SEC用来构造ADD_PACKET数据结构）；正常处理的任务在ADD POE时使用的Vqi域。（POE下发的BD中需要添加该域段）	-
-	VqiRsv	保留用作规格扩展（SEC用来构造ADD_PACKET数据结构）；正常处理的任务在ADD POE时使用的Vqi的保留域。	-
-	Location	目的POE的编码，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41。（POE下发的BD中需要添加该域段）	-
-	OrderTagValue[23:0]	24位流号，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41。（POE下发的BD中需要添加该域段）	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Affinity_TagValue[23:0]	24位亲和性标识，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41。（POE下发的BD中需要添加该域段）	-
-	Order_Type	保序类型，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41：（POE下发的BD中需要添加该域段）	-
-		0x0: NULL;	-
-		0x1: ORDER;	-
-		0x2: ATOMIC serial;	-
-		0x3: ATOMIC parallel。	-
-	Order_Reserved	保留用作规格扩展，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41。（POE下发的BD中需要添加该域段）	-
-	Affinity	亲和性调度指示字段，SEC用来构造ADD_PACKET数据结构，域的详细说明见POE的ADD_PACKET数据结构 0x41；（POE下发的BD中需要添加该域段）	-
-	说明		-
-	通用配置		-
-	BD_Type	数据结构类型编号：	-
-		0x1: 存储上下盘数据的重删/加解密场景；（兼容1620）	-
-		0x2: 除存储重删及上下盘加解密以外的场景；（兼容1620）	-
-		0x3: 309a/1630使用；	-
-		其他：保留；	-
-	BD_VLD	BD无效指示：	-
-		0: 该BD有效；	-
-		1: 该BD无效。	-
-	Scene	应用场景：	-
-		0x0: 无场景；	-
-		0x1: IPSEC（主控传输）；	-
-		0x2: 基带（LTE、5G）（本版不使用）；	-
-		0x3: SSL/TLS；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x4: DTLS;	-
-		0x5: 存储上下盘数据; (本版不使用);	-
-		0x6: 存储NAS传输;	-
-		0x7: 流式数据处理;	-
-		0x8: PBKDF2;	-
-		0x9: 存储SMB;	-
-		0xA: SSL 1.3/TLS 1.3;	-
-		其他: 保留。	-
-	DE	目标地址使能位;	-
-		0x0: 不使能, 加解密处理后输出数据覆盖原地址, 即加密情况下密文覆盖明文, 解密情况下明文覆盖密文;	-
-		0x1: 使能, 加解密处理后输出数据写到BD中软件通过“xx_Dst_Addr”域设定的目的地址中;	-
-		其他: 保留。	-
-	Src_Addr_Type	源数据Addr类型:	-
-		0x0: RefltslfAddr是直接地址 (单BUF结构, 此时只识别链表描述符的RefltslfAddr、DataOffset、Dsc_Len、T_TagValue、T_Len域, 忽略链表描述符的其他域段);	-
-		0x1: RefltslfAddr是SGL结构;	-
-		0x2: Data_Src_Addr/ DIF_Src_Addr是PRP结构;	-
-		0x3: PPE链式结构;	-
-		其他: 保留;	-
-	Dst_Addr_Type	目的数据Addr类型:	-
-		0x0: RefltslfAddr是直接地址 (单BUF结构);	-
-		0x1: RefltslfAddr是SGL结构;	-
-		0x2: RefltslfAddr是PRP结构;	-
-		0x3: PPE链式结构;	-
-		其他: 保留;	-
-	MAC_Addr_Type	MAC写出Addr类型:	-
-		0x0: MAC_Addr是直接地址 (单BUF结构);	-
-		0x1: MAC_Addr是SGL结构; 注意, 需要软件做数据结构转换处理;	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x2: MAC_Addr是PRP结构；注意，需要软件做数据结构转换处理；	-
-		其他：保留；	-
-	Data_Src_Addr	用来读入数据的地址，地址含义参照Src_Addr_Type配置，直接地址支持按照字节对齐，其他地址类型支持按照Cacheline对齐。	-
-	Data_Dst_Addr	用来写出数据的地址，地址含义参照Dst_Addr_Type配置，直接地址支持按照字节对齐，其他地址类型支持按照Cacheline对齐；是否使用该地址受“DE”等相关于域控制。	-
-	Cipher_IVIN_Addr	当用于非基带场景且加密且IV_GEN等于1时，为组成IVIN的随机数的写出地址；当用于其他非基带应用场景时，为加解密IVIN存放的起始地址；支持按字节对齐；	-
-	Cipher_Key_Addr	加解密密钥存放起始地址，支持按字节对齐。	-
-	Auth_IVIN_Offset	用来表示Auth_IVIN_ADDR相对Cipher_IVIN_Addr的偏移，单位为128Byte；例如：Auth_IVIN_ADDR = Cipher_IVIN_Addr + {Auth_IVIN_Offset, 7h0}；	-
-	Auth_Key_Offset	用来表示Auth_Key_ADDR相对Cipher_Key_Addr的偏移，单位为128Byte；例如：Auth_Key_ADDR = Cipher_Key_Addr + {Auth_Key_Offset, 7h0}；	-
-	Auth_Key_Addr	认证密钥存放起始地址，按字节对齐。当BD中存在Auth_Key_Addr域段时，Auth_Key_Offset无效；	-
-	Auth_IVIN_Addr	认证IVIN存放起始地址，按字节对齐。当BD中存在Auth_IVIN_Addr域段时，Auth_IVIN_Offset无效；	-
-	Cipher	加解密使能信息：	-
-		0x0：不做加解密算法；	-
-		0x1：加密；	-
-		0x2：解密；	-
-		0x3：报文硬件拷贝。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Auth	认证使能信息：	-
-		0x0：不做认证；	-
-		0x1：SEC认证的MAC值写到外部DDR/Cache中；	-
-		0x2：SEC认证的MAC值与指定地址的MAC值进行校验；	-
-		0x3：保留。	-
-	CI_G EN	加解密IVIN的获取方式：	-
-		0x0：从Cipher_IVIN_Addr读取加解密算法所需的IVIN（软件等拼好的）；	-
-		0x1：加密方向从TRNG取得组成IVIN所需的随机数，并将随机数写出至相应地址处，解密方向从相应地址处读取随机数，用以组成IVIN；所操作的地址的参见Cal_IV_Addr_En描述	-
-		0x2：需要使用Tls_1p3_gcm_ivin当做IVIN值（只有TLS1.3场景GCM算法才可能使用，TLS1.3流式数据处理场景不支持），需要使用Bearer_ID、Direction、SEC_COUNT合成IVIN值（只有基带场景下才可能选用）；	-
-		0x3：保留；	-
-	AI_G EN	认证IVIN的获取方式：	-
-		0x0：使用内置的IVIN（非流模式场景，除AES-GMAC以外的认证算法只可以选用此方式）；	-
-		0x1：从Auth_IVIN_Addr读取需要的完整的IVIN；	-
-		0x2：需要使用Bearer_ID、Direction、SEC_COUNT合成IVIN值（只有基带场景下才可能选用）；认证方向从相应地址处读取随机数，用以组成IVIN；所操作的地址的参见Cal_IV_Addr_En描述；（只有AES-GMAC算法才可能选用）	-
-		0x3：认证方向从TRNG取得组成IVIN所需的随机数，并将随机数写出至相应地址处，解认证方向相应地址处读取随机数，用以组成IVIN；所操作的地址的参见Cal_IV_Addr_En描述；（只有AES-GMAC算法才可能选用）；	-
-	SEQ	加解密和认证的处理顺序：	-
-		0：先做加解密再做认证；	-
-		1：先做认证再做加解密。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	{C_Alg, C_Mode, CKey_Len, C_Width, C_ICV_Len}	C_Alg: 加解密算法类型:	-
-		0x0: DES;	-
-		0x1: 3DES;	-
-		0x2: AES;	-
-		0x3: SM4;	-
-		0x4: SNOW3G;	-
-		0x5: ZUC;	-
-		其他: 保留。	-
-		C_Mode: 加解密算法模式:	-
-		0x0: ECB;	-
-		0x1: CBC;	-
-		0x4: CTR;	-
-		0x3: OFB;	-
-		0x5: CCM;	-
-		0x6: GCM;	-
-		0x7: XTS	-
-		0x8: SNOW3G/ZUC。	-
-		0x9: CBC_CS	-
-		CKey_Len: Cipher运算密钥长度控制（在XTS模式下表示Key1或Key2的长度）:	-
-		0x0: AES、SM4、SNOW3G、ZUC运算时，表示128位密钥长度;	-
-		0x1: AES运算时，表示192位密钥长度; 3DES运算时，表示3个密钥; DES运算时，表示1个64位密钥;	-
-		0x2: AES运算时，表示256位密钥长度;	-
-		0x3: 3DES运算时，表示2个密钥。	-
-		其他: 保留。	-
-		C_Width: 在CBC_CS模式下，为CBC-CS类型指示（此时配置范围为0x1~0x3），其他模式下为Cipher位宽控制（此时C_Width域必须配置为0x0）	-
-		C_ICV_Len: CCM、GCM算法下ICV长度，单位: Byte;	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作	-
-		-
-	CCM、GCM包含机密性和完整性计算，若配置为CCM、GCM模式则忽略{A_Alg, AKey_Len, MAC_Len}的配置；	-
-	加解密算法 {C_Alg, C_Mode, CKey_Len, C_Width, C_ICV_Len} 配置指引如下：	-
-	0x0_0200 (20' b0000_0000_001_000_000000)：DES-ECB-64；	-
-	0x0_1200 (20' b0000_0001_001_000_000000)：DES-CBC-64；	-
-	0x1_0200 (20' b0001_0000_001_000_000000)：3DES-ECB-192（3个密钥）；	-
-	0x1_0600 (20' b0001_0000_011_000_000000)：3DES-ECB-192（2个密钥）；	-
-	0x1_1200 (20' b0001_0001_001_000_000000)：3DES-CBC-192（3个密钥）；	-
-	0x1_1600 (20' b0001_0001_011_000_000000)：3DES-CBC-192（2个密钥）；	-
-	0x2_0000 (20' b0010_0000_000_000_000000)：AES-ECB-128；	-
-	0x2_0200 (20' b0010_0000_001_000_000000)：AES-ECB-192；	-
-	0x2_0400 (20' b0010_0000_010_000_000000)：AES-ECB-256；	-
-	0x2_1000 (20' b0010_0001_000_000_000000)：AES-CBC-128；	-
-	0x2_1200 (20' b0010_0001_001_000_000000)：AES-CBC-192；	-
-	0x2_1400 (20' b0010_0001_010_000_000000)：AES-CBC-256；	-
-	0x2_4000 (20' b0010_0100_000_000_000000)：AES-CTR-128；	-
-	0x2_4200 (20' b0010_0100_001_000_000000)：AES-CTR-192；	-
-	0x2_4400 (20' b0010_0100_010_000_000000)：AES-CTR-256；	-
-	0x2_50xx (20' b0010_0101_000_000_xxxxxx)：AES-CCM-128（CCM C_ICV_Len可配置值为：0x4、0x6、0x8、0xA、0xC、0xE、0x10）；	-
-	0x2_52xx (20' b0010_0101_001_000_xxxxxx)：AES-CCM-192（CCM C_ICV_Len可配置值为：0x4、0x6、0x8、0xA、0xC、0xE、0x10）；	-
-	0x2_54xx (20' b0010_0101_010_000_xxxxxx)：AES-CCM-256（CCM C_ICV_Len可配置值为：0x4、0x6、0x8、0xA、0xC、0xE、0x10）；	-
-	0x2_60xx (20' b0010_0110_000_000_xxxxxx)：AES-GCM-128（GCM C_ICV_Len可配置值为：0x8、0x9、0xA、0xB、0xC、0xD、0xE、0xF、0x10）；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作	-
-		-
-	0x2_62xx (20' b0010_0110_001_000_00000000) : AES-GCM-192 (GCM C_ICV_Len可配置值为: 0x8、0x9、0xA、0xB、0xC、0xD、0xE、0xF、0x10) ;	-
-	0x2_64xx (20' b0010_0110_010_000_00000000) : AES-GCM-256 (GCM C_ICV_Len可配置值为: 0x8、0x9、0xA、0xB、0xC、0xD、0xE、0xF、0x10) ;	-
-	0x2_7000 (20' b00010_0111_000_000_00000000) : AES-XTS-128;	-
-	0x2_7400 (20' b0010_0111_010_000_00000000) : AES-XTS-256;	-
-	0x2_9040 (20' b0010_1001_000_001_00000000) : AES-CBC-CS1-128;	-
-	0x2_9240 (20' b0010_1001_001_001_00000000) : AES-CBC-CS1-192;	-
-	0x2_9440 (20' b0010_1001_010_001_00000000) : AES-CBC-CS1-256;	-
-	0x2_9080 (20' b0010_1001_000_010_00000000) : AES-CBC-CS2-128;	-
-	0x2_9280 (20' b0010_1001_001_010_00000000) : AES-CBC-CS2-192;	-
-	0x2_9480 (20' b0010_1001_010_010_00000000) : AES-CBC-CS2-256;	-
-	0x2_90C0 (20' b0010_1001_000_011_00000000) : AES-CBC-CS3-128 (CTS模式) ;	-
-	0x2_92C0 (20' b0010_1001_001_011_00000000) : AES-CBC-CS3-192 (CTS模式) ;	-
-	0x2_94C0 (20' b0010_1001_010_011_00000000) : AES-CBC-CS3-256 (CTS模式) ;	-
-	0x2_3000 (20' b0010_0011_000_000_00000000) : AES-OFB-128;	-
-	0x2_3200 (20' b0010_0011_001_000_00000000) : AES-OFB-192;	-
-	0x2_3400 (20' b0010_0011_010_000_00000000) : AES-OFB-256;	-
-	0x3_1000 (20' b0011_0001_000_000_00000000) : SM4-CBC-128;	-
-	0x3_4000 (20' b0011_0100_000_000_00000000) : SM4-CTR-128;	-
-	0x3_7000 (20' b0011_0111_000_000_00000000) : SM4-XTS-128;	-
-	0x3_3000 (20' b0011_0011_000_000_00000000) : SM4-OFB-128;	-
-	其他: 保留。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	{A_Alg, AKey_Len, MAC_Len}	A_Alg: 认证算法类型。	-
-		AKey_Len: 认证密钥长度，单位：Word。	-
-		MAC_Len: 认证MAC长度，单位：Word。	-
-		认证算法 {A_Alg, AKey_Len, MAC_Len} 配置指引如下：	-
-		0x0_00xx (17' b0000000_0000000_xxxxx)：SHA-1（该算法MAC值长度为160bit，允许的编码空间为0x0~0x5，其中配置为0表示要输出全长，配置效果等同于配置0x5）；	-
-		0x0_08xx (17' b0000001_0000000_xxxxx)：SHA-256（该算法MAC值长度为256bit，允许的编码空间为0x0~0x8，其中配置为0表示要输出全长，配置效果等同于配置0x8）；	-
-		0x0_10xx (17' b0000010_0000000_xxxxx)：MD5（该算法MAC值长度为128bit，允许的编码空间为0x0~0x4，其中配置为0表示要输出全长，配置效果等同于配置0x4）；	-
-		0x0_18xx (17' b0000011_0000000_xxxxx)：SHA-224（该算法MAC值长度为224bit，允许的编码空间为0x0~0x7，其中配置为0表示要输出全长，配置效果等同于配置0x7）；	-
-		0x0_20xx (17' b0000100_0000000_xxxxx)：SHA-384（该算法MAC值长度为384bit，允许的编码空间为0x0~0xC，其中配置为0表示要输出全长，配置效果等同于配置0xC）；	-
-		0x0_28xx (17' b0000101_0000000_xxxxx)：SHA-512（该算法MAC值长度为512bit，允许的编码空间为0x0~0x10，其中配置为0表示要输出全长，配置效果等同于配置0x10）；	-
-		0x0_30xx (17' b0000110_0000000_xxxxx)：SHA-512/224（该算法MAC值长度为224bit，允许的编码空间为0x0~0x7，其中配置为0表示要输出全长，配置效果等同于配置0x7）；	-
-		0x0_38xx (17' b0000111_0000000_xxxxx)：SHA-512/256（该算法MAC值长度为256bit，允许的编码空间为0x0~0x8，其中配置为0表示要输出全长，配置效果等同于配置0x8）；	-
-		0x0_80xx (17' b0100000_xxxxxx_xxxxx)：HMAC-SHA1（该算法Key的长度为0~512bit，允许的编码空间为0x0~0x10；该算法MAC值长度为160bit，允许的编码空间为0x1~0x5）；	-
-		0x0_88xx (17' b0100001_xxxxxx_xxxxx)：HMAC-SHA256（该算法Key的长度为0~512bit，允许的编码空间为0x0~0x10；该算法MAC值长度为256bit，允许的编码空间为0x1~0x8）；	-
-		0x0_90xx (17' b0100010_xxxxxx_xxxxx)：HMAC-MD5（该算法Key的长度为0~512bit，允许的编码空间为0x0~0x10；该算法MAC值长度为128bit，允许的编码空间为0x1~0x4）；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x0_98xx (17' b010011_XXXXXX_XXXXX) : HMAC-SHA224 (该算法Key的长度为0~512bit, 允许的编码空间为0x0~0x10 ; 该算法MAC值长度为224bit, 允许的编码空间为0x1~0x7) ;	-
-		0x0_A0xx (17' b010100_XXXXXX_XXXXX) : HMAC-SHA384 (该算法Key的长度为0~1024bit, 允许的编码空间为0x0~0x20 ; 该算法MAC值长度为384bit, 允许的编码空间为0x1~0xC) ;	-
-		0x0_A8xx (17' b010101_XXXXXX_XXXXX) : HMAC-SHA512 (该算法Key的长度为0~1024bit, 允许的编码空间为0x0~0x20 ; 该算法MAC值长度为512bit, 允许的编码空间为0x1~0x10) ;	-
-		0x0_B0xx (17' b010110_XXXXXX_XXXXX) : HMAC-SHA512/224 (该算法Key的长度为0~1024bit, 允许的编码空间为0x0~0x20 ; 该算法MAC值长度为224bit, 允许的编码空间为0x1~0x7) ;	-
-		0x0_B8xx (17' b010111_XXXXXX_XXXXX) : HMAC-SHA512/256 (该算法Key的长度为0~1024bit, 允许的编码空间为0x0~0x20 ; 该算法MAC值长度为256bit, 允许的编码空间为0x1~0x8) ;	-
-		0x1_0083 (17' b100000_000100_00011) : AES-XCBC-MAC-96 ;	-
-		0x1_0084 (17' b100000_000100_00100) : AES-XCBC-PRF-128 ;	-
-		0x1_088x (17' b100001_000100_XXXXX) : AES-CMAC (该算法MAC值长度为128bit, 允许的编码空间为0x1~0x4) ;	-
-		0x1_10xx (17' b100010_XXXXXX_XXXXX) : AES-GMAC (该算法Key的长度为128bit, 192bit, 256bit, 允许的编码为0x4、0x6、0x8 ; 该算法MAC值长度为128bit, 允许的编码空间为0x1~0x4)	-
-		0x1_28xx (17' b100101_000000_XXXXX) : SM3 (该算法MAC值长度为256bit, 允许的编码空间为0x1~0x8) ;	-
-		0x1_30xx (17' b100110_XXXXXX_XXXXX) : HMAC-SM3 (该算法Key的长度为0~512bit, 允许的编码空间为0x0~0x10 ; 该算法MAC值长度为256bit, 允许的编码空间为0x1~0x8) ;	-
-		其他：保留。	-
-	Non ce_L en	CI_GEN域等于1时，用于指示AES-CCM算法下Nonce的长度，需要固定配置为11。	-
-	HUK	HUK使用标志，安全队列下的任务，该bit配置有效；非安全队列下的任务，忽略该配置，固定识别为0	-
-		0：使用BD中对应Key地址读入的Key完成运算；	-
-		1：使用HUK完成运算（加解密和认证算法都使用HUK）。	-
-	KEY_ SEL	KEY_SEL密钥选择。	-
-		0：非安全任务从DDR读取明文KEY；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		1: HUK_DER，安全任务才有效；	-
-		2: IMG_KEY1，安全任务才有效；（309a 项目无效）	-
-		3: IMG_KEY2，安全任务才有效；（309a 项目无效）	-
-		4: WRAPPING_KEY，安全和非安全任务都有效；	-
-		5: STROAGE_KEY，安全任务才有效；	-
-		6: 保留；	-
-		7: 保留；	-
-		8: EFUSE_KEY1，安全任务才有效；	-
-		9: EFUSE_KEY2，安全任务才有效；	-
-		其他: 保留；	-
-	Cal_IV_Addr_En	是否使用通过IV_Offset计算出的地址读或写组成IVIN的随机数部分：	-
-		0: 读地址为Cipher_IVIN_Addr或者Auth_IVIN_Addr，回写地址为两处（第一处地址使用待处理数据首地址+IV_Offset，第二处地址使用Cipher_IVIN_Addr或者Auth_IVIN_Addr）；	-
-		1: 读写地址为同一处（地址使用待处理数据首地址+IV_Offset）（IPSEC场景下才允许配置为1）。	-
-	Updata_Key	当KEY_SEL配置为WRAPPING_KEY或STROAGE_KEY时，是否强制重新从外部中读取新数据：	-
-		0: 如果SEC中已经存在这个KEY，则使用已有KEY；否则从外部读取新的数据解密出KEY；	-
-		1: 从外部读取数据解密出KEY；	-
-	Salt	CI_GEN域等于1且执行AES-CCM/AES-GCM/ AES-CTR算法时用来与随机数一起组成Nonce，AES-CCM时长度为3 Byte（salt[23:0]），AES-GCM时长度为4 Byte，AES CTR时长度为4 Byte。	-
-	Auth_Src_Offset	认证报文部分到原报文头读入源地址的偏移地址，按照字节对齐。	-
-	Cipher_Src_Offset	加解密报文部分到原报文头读入源地址的偏移地址，按照字节对齐。	-
-	Auth_Len	认证报文长度；当执行AES-CCM和AES-GCM算法时，表示AAD数据的长度；单位：Byte。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Cipher_Len	加解密报文长度，单位：Byte。AES和SM4的ECB、CBC模式支持到16Byte对齐，DES/3DES的ECB、CBC模式支持到8Byte对齐，其他算法（模式）支持到字节对齐。	-
-	MAC_Addr	认证处理后MAC写出地址/认证结果MAC值与该地址的数据进行匹配，支持按照字节对齐。	-
-	Auth_Key_Addr	认证密钥存放起始地址，按字节对齐。	-
-	Auth_IVIN_Addr	认证IVIN存放起始地址，按字节对齐。	-
-	IPSEC		-
-	C_S	IPSEC场景指示是否用硬件处理IPv6 UDP Checksum（SEC不做报文头类型识别，软件保证只在适当场景下且IPv6报文类型才将该域配成1或者2）：	-
-		0x0：不进行IPv6 UDP Checksum处理；	-
-		0x1：计算IPv6 UDP Checksum；	-
-		0x2：校验IPv6 UDP Checksum；	-
-		0x3：保留；	-
-	Deal_ESP_AH	IPSEC场景下有效：SEC对ESP、AH协议封装的处理方式：	-
-		0x0：SEC不对ESP、AH协议封装做处理；	-
-		0x1：SEC先添加ESP Tail后再对报文做加密认证等处理（不需要更改IP头，要求软件给SEC前已处理好；需要特别注意的是：Cipher_Len是含ESP Tail的待加密数据总长，Auth_Len是含ESP Tail的待认证数据总长）；	-
-		0x2~0x4：保留；	-
-		0x4：SEC对报文做完解密认证等处理后剥除ESP、AH协议封装（含头尾、含外层头），并相应修改IP头和链表头；	-
-		0x5~0x7：保留；	-
-	Protocol_Type	IPSEC场景，加解密认证协议类型：（建议都配置为0）	-
-		0x0：所有报文；	-
-		0x1：AH；	-
-		其他：保留；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Mode	报文模式类型：	-
-		0x0：传输模式；	-
-		0x1：隧道模式；	-
-		其他：保留；	-
-	IP_Type	IP报文协议版本：	-
-		0x0：IPv4；	-
-		0x1：IPv6；	-
-		其他：保留；	-
-	Padding Length	IPSEC场景下有效：ESP Tail的Padding Length域，报文后所加padding的长度，单位：Byte。	-
-	Next Header	IPSEC场景下有效：ESP Tail的Next Header域，上层协议头。	-
-	IV_Offset	IV部分相对于原报文头读入源地址的偏移地址，按照字节对齐。	-
-	CS_IP_Header_Offset	IPSEC场景下做Check Sum计算所用的IP头（内层头）相对于原报文头读入源地址的偏移，按照字节对齐。	-
-	CS_UDP_Header_Offset	IPSEC场景下做Check Sum计算所用的UDP头相对于原报文头读入源地址的偏移，按照字节对齐。	-
-	SSL/TLS/DTLS		-
-	Cipher_Pad_Type	SSL/TLS对待处理的数据进行padding操作的类型（Cipher_Pad_Type配置值识别成0x0时，Cipher_Len包含所有待加解密数据的长度；Cipher_Pad_Type非0时，Cipher_Len不包含padding和pad_len域部分）：	-
-		0x0：不做padding；	-
-		0x1：PKCS5 Padding（补齐成block对齐，如3DES-CBC，数据长19B，则padding内容为 05 05 05 05 05；如果数据长24B，则padding内容为 08 08 08 08 08 08 08）（选择加密方向块加密算法时配置有效，否则识别成不做padding）；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x2: TLS padding算法且最短合法填充长度；	-
-		0x3: 软件指示Padding数据长度和内容（此时Cipher_Pad_Len才会有效，注意其不用Pad_Data_Type、Pad_Len_Field配置）；	-
-		其他：保留（识别成不做padding）。	-
-	Cipher_Pad_Data_Type	SSL/TLS每个字节所padding数据的内容：（只能配置为0）	-
-		0x0: 0；	-
-		0x1: 保留；	-
-		0x2: 递增（1~255）；	-
-		0x3: Pad_Len；	-
-		其他：保留。	-
-	Cipher_Pad_Len	SSL/TLSpadding数据的长度，单位：Byte	-
-	Cipher_Pad_Len_Field	SSL/TLS是否在padding数据后附上Pad_Len域（长度为1Byte）：（只能配置为0）	-
-		0x0: 不在padding数据后附上Pad_Len域；	-
-		0x1: 在padding数据后附上Pad_Len域，内容为Pad_Len；	-
-		0x2: 在padding数据后附上Pad_Len域，内容为Pad_Len+1；	-
-		0x3: 保留。	-
-	SN	SSL/TLS场景下的Sequence Number，用来参与认证运算。	-
-	存储		-
-	Gran_Num	数据粒度数（扇区数），加解密和认证共享这个域，也就是说加解密和认证的Gran数是一致的。 需要注意的是，存储场景下，AES-CCM/AES-GCM算法仅支持Gran_Num为1的情况。	-
-	Src_Skip_Data_Length	存储场景下，支持（从Data_Src_Addr开始）跳过不做处理的数据长度，单位：Byte。（只有pbuf和SGL链时支持，PRP链时不支持）	-
-	Dst_Skip_Data_Length	存储场景下，支持（从Data_Dst_Addr开始）跳过不做处理的数据长度，单位：Byte。（只有pbuf和SGL链时支持，PRP链时不支持）	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Gen_ VER_ Val	DIF中的Version域如果采用系统输入时的值，位宽8bit； 8h13：压缩、解压、加密、解密典型值；EC/RAID（D） 8h34：EC/RAID（P/Q）（SEC不会使用）； 其他：保留。	-
-	Gen_ APP_ Val	DIF中的APP域如果采用系统输入时的值，位宽8bit；	-
-	Gen_ Page_ Pad	表示是否生成56Byte的Page Pad，如何生成，位宽4bit；	-
-	Ctrl	0x0：不生成Pad；	-
-		0x3：生成Pad，默认为0；	-
-		0x4：生成Pad，从源中继承；	-
-		0x5：生成Pad，EC/RAID算法生成；（SEC不会使用）；	-
-		其他：保留。	-
-	Gen_ GRD_ Ctrl	表示是否生成DIF中的GRD域，如何生成，位宽4bit；	-
-		0x0：不生成GRD；	-
-		0x1：生成GRD，采用T10 CRC方式；	-
-		0x5：生成GRD，EC/RAID算法生成；（SEC不会使用）；	-
-		其他：保留。	-
-	Gen_ VER_ Ctrl	表示是否生成DIF中的Version域，如何生成，位宽4bit：	-
-		0x0：不生成Version；	-
-		0x1：生成Version，采用系统输入的VER_Gen_Val值；	-
-		0x3：生成Version，默认为0；	-
-		0x4：生成Version，从源中继承。	-
-		其他：保留。	-
-	Gen_ APP_ Ctrl	表示是否生成DIF中的APP域，如何生成，位宽4bit：	-
-		0x0：不生成APP；	-
-		0x1：生成APP，采用系统输入的APP_Gen_Val值；	-
-		0x3：生成APP，默认为0；	-
-		0x4：生成APP，从源中继承。	-
-		0x5：生成APP，EC/RAID算法生成（SEC不会使用）。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		其他：保留。	-
-	Gen_REF_Ctrl	表示是否生成REF域，如何生成，位宽4bit。	-
-		0x0：不生成REF；	-
-		0x1：生成REF，第一个block采用系统输入的LBA域的低4个Byte，后续每个block+1；	-
-		0x2：生成REF，第一个block采用系统输入的Gen_Private_Info，后续每个block+1；	-
-		0x3：生成REF，默认为0；	-
-		0x4：生成REF，从源中继承；	-
-		0x5：生成REF，EC/RAID算法生成；	-
-		其他：保留。	-
-	Page_Pad_Type	表示输入和输出是否包含56Byte的Page Pad，以及位置，输入输出共用：	-
-		0x0：无Pad；	-
-		0x1：有Pad，在DIF前；	-
-		0x2：有Pad，在DIF后；	-
-		其他：保留。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Block_Size	<p>表示每个扇区的大小，不含DIF，单位是Byte，输入和输出相同，共用此域，位宽16bit；</p> <p>常用的Block_Size有512、520、4096、4160、4104五种。</p> <p>Block_Size和Page_Pad_Type，Gen_GRD_Ctrl一起表示输出数据的DIF格式：</p> <ol style="list-style-type: none">1. 512无DIF：Block_Size=512，Gen_GRD_Ctrl = 0；2. 512+8：Block_Size=512，Page_Pad_Gen_Type=0，Gen_GRD_Ctrl=1；3. 520无DIF：Block_Size=520，Gen_GRD_Ctrl = 0；4. 4096无DIF：Block_Size=4096，Gen_GRD_Ctrl = 0；5. 4096+8：Block_Size=4096，Page_Pad_Gen_Type=0，Gen_GRD_Ctrl = 1；6. 4096+56（偏移）+8：Block_Size=4096，Page_Pad_Gen_Type=1，Gen_GRD_Ctrl=1；7. 4096+8+56（偏移）：Block_Size=4096，Page_Pad_Gen_Type=2，Gen_GRD_Ctrl=1；8. 4160无DIF：Block_Size=4160，Gen_GRD_Ctrl = 0；9. 4104无DIF：Block_Size=4104，Gen_GRD_Ctrl = 0。 <p>Block_Size和Page_Pad_Type，Chk_GRD_Ctrl一起表示输入数据的DIF格式：</p> <ol style="list-style-type: none">1. 512无DIF：Block_Size=512，Chk_GRD_Ctrl = 0；2. 512+8：Block_Size=512，Page_Pad_Type = 0，Chk_GRD_Ctrl != 0；3. 520无DIF：Block_Size=520，Chk_GRD_Ctrl = 0；4. 4096无DIF：Block_Size=4096，Chk_GRD_Ctrl = 0；5. 4096+8：Block_Size=4096，Page_Pad_Type = 0，Chk_GRD_Ctrl != 0；6. 4096+56（偏移）+8：Block_Size=4096，Page_Pad_Type = 1，Chk_GRD_Ctrl != 0；7. 4096+8+56（偏移）：Block_Size=4096，Page_Pad_Type = 2，Chk_GRD_Ctrl != 0；8. 4160无DIF：Block_Size=4160，Chk_GRD_Ctrl = 0；9. 4104无DIF：Block_Size=4104，Chk_GRD_Ctrl = 0。 <p>注意：在一个任务中，输入的格式和输出的格式要保持一致。</p>	-
-	Gen_Private_Info	<p>DIF域生成时，REF字段第一个block地址采用的系统私有信息，长度为4Byte。</p>	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Chk_GRD	表示对DIF中GRD域是否存在，对其是否校验，位宽4bit：	-
-	-	0x0：无GRD；	-
-	Ctrl	0x1：有GRD，不校验；（注意：Chk_GRD_Ctrl作为GRD和REF是否校验的总开关，即Chk_GRD_Ctrl = 1不校验时，Chk_REF_Ctrl也不会校验，如果想要校验DIF，需要把Chk_GRD_Ctrl配成2）	-
-		0x2：有GRD，校验（T10 CRC）；	-
-		其他：保留。	-
-	Chk_REF_Ctrl	表示DIF中REF域是否存在，是否校验，以及如何校验，位宽4bit：	-
-		0x0：无REF；	-
-		0x1：有REF，不校验；	-
-		0x2：有REF，对于每个block都进行校验，初始值为输入LBA的低4个Byte，后续每扇区+1；	-
-		0x3：有REF，对于每个block都进行校验，初始值为输入LBA的低4个Byte，后续每扇区+1；	-
-		其他：保留。	-
-	流式数据		-
-	Auth_Pad	流模式场景GCM算法和长数据处理时，与AI_GEN组合指示首中尾BD：	-
-		0x0：流模式GCM算法，同时AI_GEN = 1，指示尾间BD； 长数据HASH场景，同时AI_GEN = 1，Auth_Pad = 0指示尾BD；	-
-		0x1：流模式GCM算法，同时AI_GEN = 0，指示非尾BD； 长数据HASH场景，同时AI_GEN = 1，指示首BD或中间BD；	-
-		其他：保留；	-
-	Long_Auth_Data_Len	长数据Hash尾部（即AI_GEN=1 Auth_Pad_Type=0）时，表示Hash做PAD时需要填充的长度域，PAD里长度域的值等于整个长数据Hash处理的报文的总长度，以bit为单位，SEC默认该域低3bit为0，不做检查。	-
-	Stream_Protocol	流式数据处理场景（Scene=0x7）时，适配ESP、TLS1.3填Padding的情况：	-
-		0x0：不填Padding；（如TLS1.2）	-
-		0x1：ESP协议加密方向时，填Padding指示（只在尾BD有效）；	-
-		0x2：TLS1.3协议GCM算法加密方向，填Type填Padding；（Type使用Plaintext_Type域段的值，Padding长度由Padding_Length_1p3指示）	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x3: TLS1.3协议GCM算法加密方向，不填Type填Padding； (Padding长度由Padding Length_1p3指示)	-
-		0x4: TLS1.3协议GCM算法解密方向，提取Type和Padding长度；(回 写到Tls_1p3_Type_Back和Padding Length_1p3_Back)	-
-		其他：保留；	-
-	Padd ing Leng th_1 p3	流式处理场景，TLS 1.3协议下有效：加密方向添加Padding的长度。单位：Byte。	-
-	Plain text_ Type	流式处理场景，TLS 1.3协议下有效：加密方向填写Type的值。	-
-	Padd ing Leng th	流式处理场景：当Stream_Protocol为0x1时有效：ESP Tail的Padding Length域，报文后所加padding的长度，单位：Byte。	-
-	Next Hea der	流式处理场景：当Stream_Protocol为0x1时有效：ESP Tail的Next Header域，上层协议头。	-
-	PBKDF2		-
-	PBK DF2_ Salt_ Len	做密钥派生时输入的Salt（密码学意义上的安全随机数，由系统随机生成）的长度，单位：Byte，至少为8。（在PBKDF2场景下，通用配置中的Auth_len需要跟此值保持一致）	-
-	Pass _Wo rd_L en	做密钥派生时输入的PassWord的长度，单位：Byte，当前产品应用范围为8~32，典型值为8。	-
-	C_N um	做密钥派生时hash算法的迭代次数，典型值为10000。	-
-	DK_L en	做密钥派生时输出派生密钥的长度，单位：Byte，典型值为32。	-
-	TLS1.3		-
-	Padd ing Leng th_1 p3	TLS 1.3场景下有效：加密方向添加Padding的长度。单位：Byte。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-	Plain text_Type	TLS 1.3场景下有效：加密方向填写Type的值。	-
-	Deal_TLS_1p3	TLS 1.3场景下有效：加密方向PayLoad后添Type和填Padding的模式：	-
-		0x0：硬件不填Type，不填padding；	-
-		0x1：硬件不填Type，填Padding；	-
-		0x2：硬件填Type，填Padding；	-
-		其他：保留。	-
-	Tls_1p3_gcm_ivin	CI_GEN=0x2时，TLS1.3场景GCM算法选用BD中的IVIN数据；（流式数据处理场景 不支持ci_gen=0x2） CI_GEN=0x1时，{Cipher_IVIN_Addr[63:0]，Auth_IVIN_Addr[63:0]} = Tls_1p3_gcm_ivin[127:0]；此时word6 和word7无效。	-
-	TLS_Len_Update	TLS1.2场景下是否更新TLS报文中Len域：	-
-		0：不更新Len域，保持输入值不变；	-
-		1：SEC模块更新Len域。	-
-	回写		-
-	Done	SEC处理结束标志位：	-
-		0：未处理；	-
-		1：已处理。	-
-	ICV	SEC 完整性检查ICV Check输出结果：	-
-		0x0：无完整性检查；	-
-		0x1：完整性检查通过；	-
-		0x2：完整性检查失败；	-
-		0x3：SEC BD、链表检测错误，或处理任务时遇到总线ERR；	-
-		其他：保留。	-
-	CSC	IPv6 UDP Checksum校验输出结果：	-
-		0x0：无Checksum检查；	-
-		0x1：Checksum检查通过；	-
-		0x2：Checksum检查失败。	-
-		其他：保留。	-
-	Flag	bit[0]：完整性处理标志。	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0：完整性未处理；	-
-		1：完整性已处理。	-
-		bit[1]：机密性处理标志。	-
-		0：机密性未处理；	-
-		1：机密性已处理。	-
-		bit[2]：CheckSum处理标志。	-
-		0：CheckSum未处理；	-
-		1：CheckSum已处理。	-
-	Error _Typ e	错误信息：	-
-		0x0：无错误；	-
-		0x1：读BD发生总线ERR；	-
-		0x2：BD选择了SEC不支持的算法或KEY权限错误；	-
-		0x3：BD类型选择错误或BD类型与域段不匹配等；	-
-		0x4：单个域段值非法；	-
-		0x5：BD中域段组非法；	-
-		0x6：某些域段组合错误；	-
-		0x7：IPSEC场景下的非法组合；	-
-		0x8：存储场景下的非法组合；	-
-		0x9：PBKDF2场景下的非法组合；	-
-		0xA：非流模式或长数据HASH下的非法组合；	-
-		0xB：TLS和DTLS场景下的非法组合；	-
-		0xC：流模式下的非法组合；	-
-		0xF：读KEY 发生总线异常；	-
-		0x10：读IVIN 发生总线异常；	-
-		0x11：读数据方向读链表时，发生总线异常；	-
-		0x12：读数据方向读链表时，发生链表解析错误；	-
-		0x13：读数据时，发生总线异常；	-
-		0x14：写数据方向读链表时，发生总线异常；	-
-		0x15：写数据方向读链表时，发生链表解析错误；	-



-	SEC BD结构软件可见，需要软件配置以便控制SEC进行工作		-
-			-
-		0x16: CHKSUM错误，报文中的UDP_LEN错误；	-
-		0x17: Unwrap key模块输出错误指示。	-
-		其他：保留。	-
-	Warning_Type	告警信息：	-
-		0x0: 无告警；	-
-		0x1: 有；	-
-		其他：保留；	-
-	MAC_I	SEC计算出的MAC_I结果（最低的那个word），软件根据记录的UE发送过来的MAC_I和SEC HAC计算出的MAC_I比对，用于认证校验失败时的维测；在无认证算法时输出随机值。	-
-	Checksum_I	SEC计算出的Checksum结果，软件根据记录的UE发送过来的Checksum和SEC HAC计算出的Checksum比对，用于Checksum校验失败时的维测；在无Checksum时输出随机值。	-
-	Counter	流式数据处理场景下，CTR模式软件将上一个BD中回写的Counter值填到这个域段（首BD填0）。	-
-	Reserved	保留域。	-
-	Padding_Length_1p3_Back	TLS1.3解密方向提取的Padding长度。	-
-	Tls_1p3_Type_Back	TLS1.3解密方向提取的明文Type值。	-



8.6.2 SEC 输入包任务描述符

表 8-10 SEC 输入包任务描述符

POE模式， SEC GET_POEE																															
软件可见寄存器接口（ HAC GET POE接口数据结构的基础字段）																															
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
Lo ca tio n		Vq i_ Rs v		Vqi[9:0]										SourcePort[7:0]							T_Len[3: 0] =4b0100			T_TagV alue[3: 0] =4b000 1			M sg _L en =0 b0 1				
R e s e r v e d	O r d e r _ B i n d V a l	Order_ BindID			O r d e r _ T y p e	Order_TagValue[23:0]																									
Addr[31:0]																															
R e l	R l s _ l i n d	B D	M B	Addr _Rsv	Pref_Of fset1	Pref_Of fset2	Addr[47:32]																								
Reserved																					B d _ T y p e _ S e l	T_Len[3 :0] =4b000 1			T_TagV alue[3: 0] =4b011 0 (SEC)						



POE模式，SEC GET_POEE																																		
软件可见寄存器接口（ HAC GET POE接口数据结构的基础字段）																																		
3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0													
Reserved																																		
Hac_Occ_Timestamp_Value																					T_Len[3:0] =4b0001		T_TagV alue[3:0] =4b0011 or 4b0111 or 4b1000 (Timest amp)											
Hac_Occ_CRC						Hac_Occ_ Add_ID						Hac_Occ_Ost d_SN						Hac_ Occ_ DFX_ VFID				Hac_ Occ_F RD_Q os				T_Len[3:0] =4b0001		T_TagV alue[3:0] =4b0100 (Hardw are)						
说 明																																		
POE基础字段						请参阅 SEC输入包任务描述符 、 SEC输出包任务描述符 。																												
Addr[31:0]						SEC/SA的BD地址可以使用软件预留或者由生产者自行申请BD缓存和消费者释放缓存，建议软件使用PPE包存储结构中的软件预留区。																												
Bd_Type_Sel						SEC BD TYPE 选择信号。 0：启用BD TYPE 1或2； 1：启用BD TYPE 3。																												

8.6.3 SEC 输出包任务描述符

须知

所有加速器与POE交互的数据结构长度都是256bit，包括Add和Get，数据结构定义未
满256bit的，剩余的部分Padding为0。



表 8-11 SEC 输出包任务描述符

POE模式，SEC_ADD_POE 软件可见寄存器接口（HAC_ADD_POE接口数据结构的基础字段）																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Location		Vqi_Rsv		Vqi[9:0]										SourcePort[7:0]										T_Len[3:0] =4b0100			T_TagValue[3:0] =4'b0001			Msg_Len =0b01	
Reserved	Order_BindVal	Order_BindID		Order_Type	Order_TagValue[23:0]																										
Addr[31:0]																															
Rel	Rel_Ind	BD	MB	Addr_Rsv				Pref_Offset1				Pref_Offset2				Addr[47:32]															



POE模式，SEC_ADD_POE 软件可见寄存器接口（HAC_ADD_POE接口数据结构的基础字段）											
Auth	Cipher	SEC_SGL	CSC	ICV	Hac_Occ_SEC_Err	Flag	NB	ErrorType	BD_Type_Sel	T_Len[3:0] =4b0001	T_TagValue[3:0] =4b0110 (SEC)
Reserved											
Hac_Occ_Timestamp_Value										T_Len[3:0] =4b0001	T_TagValue[3:0] =4b0011 or 4b0111 or 4b1000 (Timestamp)
Hac_Occ_CRC			Hac_Occ_Add_ID		Hac_Occ_Ost_d_SN		Hac_Occ_DFX_VFID	Hac_Occ_FRD_Qos		T_Len[3:0] =4b0001	T_TagValue[3:0] =4b0100 (Hardware)
说 明											
POE基础字段		请参阅SEC BD（SQE）。									
Addr[31:0]		SEC/SA的BD地址可以使用软件预留或者由生产者自行申请BD缓存和消费者释放缓存，建议软件使用PPE包存储结构中的软件预留区。									
SEC_SGL		SEC 回写报文是否使用SGL链式结构；0：使用单BUF；1：使用SGL链式BUF。									
Hac_Occ_SEC_Err		SEC本包处理过程中，遇到了错误（注意：Hac_Occ_SEC_Err=1时，ADD_POE时SEC使用Err_Vqi）。									
		0b0：无下述错误；									



POE模式，SEC ADD_POE 软件可见寄存器接口（HAC ADD_POE接口数据结构的基础字段）	
	0b1：SEC处理时遇到了错误。
ICV	SEC 完整性检查ICV Check输出结果。
	0x0：无完整性检查；
	0x1：完整性检查通过；
	0x2：完整性检查失败；
	0x3：SEC BD 、链表检测错误，或处理任务时遇到总线ERR。
	其他：保留。
CSC	IPv6 UDP Checksum校验输出结果。
	0x0：无Checksum检查；
	0x1：Checksum检查通过；
	0x2：Checksum检查失败；
	其他：保留。
Cipher	加解密使能信息。
	0x0：不做加解密算法；
	0x1：加密；
	0x2：解密；
	0x3：报文硬件拷贝。
Bd_Type_Sel	SEC BD TYPE 选择信号。
	0b0：启用BD TYPE 1或2；
	0b1：启用BD TYPE 3。
Auth	认证使能信息。
	0x0：不做认证；
	0x1：SEC认证的MAC值写到外部DDR/Cache中；
	0x2：SEC认证的MAC值与指定地址的MAC值进行校验；
	0x3：保留。
NB	指示SEC 新BD（BD_TYPE=3）是否搬动到新地址（同BD DE域段配置，异常或透传，忽略此域段）。
	0b0：BD原址回写；
	0b1：BD搬动到新地址。
Flag	bit[0]：完整性处理标志。



POE模式，SEC ADD_POE 软件可见寄存器接口（ HAC ADD POE接口数据结构的基础字段 ）	
	0：完整性未处理；
	1：完整性已处理。
	bit[1]：机密性处理标志。
	0：机密性未处理；
	1：机密性已处理。
	bit[2]：保留。
Error_Type	错误信息（待将来编排）。
	0x0：无错误；
	0x1：读BD发生总线ERR；
	0x2：BD选择了SEC不支持的算法或KEY权限错误；
	0x3：BD类型选择错误或BD类型与域段不匹配等；
	0x4：单个域段值非法；
	0x5：BD中域段组合非法；
	0x6：某些域段组合错误；
	0x7：IPSEC场景下的非法组合；
	0x8：存储场景下的非法组合；
	0x9：PBKDF2场景下的非法组合；
	0xA：非流模式或长数据HASH下的非法组合；
	0xB：TLS和DTLS场景下的非法组合；
	0xC：流模式下的非法组合；
	0xF：读KEY 发生总线异常；
	0x10：读IVIN 发生总线异常；
	0x11：读数据方向读链表时，发生总线异常；
	0x12：读数据方向读链表时，发生链表解析错误；
	0x13：读数据时，发生总线异常；
	0x14：写数据方向读链表时，发生总线异常；
	0x15：写数据方向读链表时，发生链表解析错误；
	0x16：CHKSUM错误，报文中的UDP_LEN错误；



POE模式，SEC ADD_POE 软件可见寄存器接口（HAC ADD_POE接口数据结构的基础字段）	
	0x17: Unwrap key模块输出错误指示；
	其他：保留。

8.6.4 数据访问方式

加速器根据数据存放方式来获取数据。数据存放的方式分为三种，分别是Pbuffer形式，SGL形式以及PPE List形式，三种都是支持DIF的生成和校验配置的。

8.6.4.1 Pbuffer 形式访问

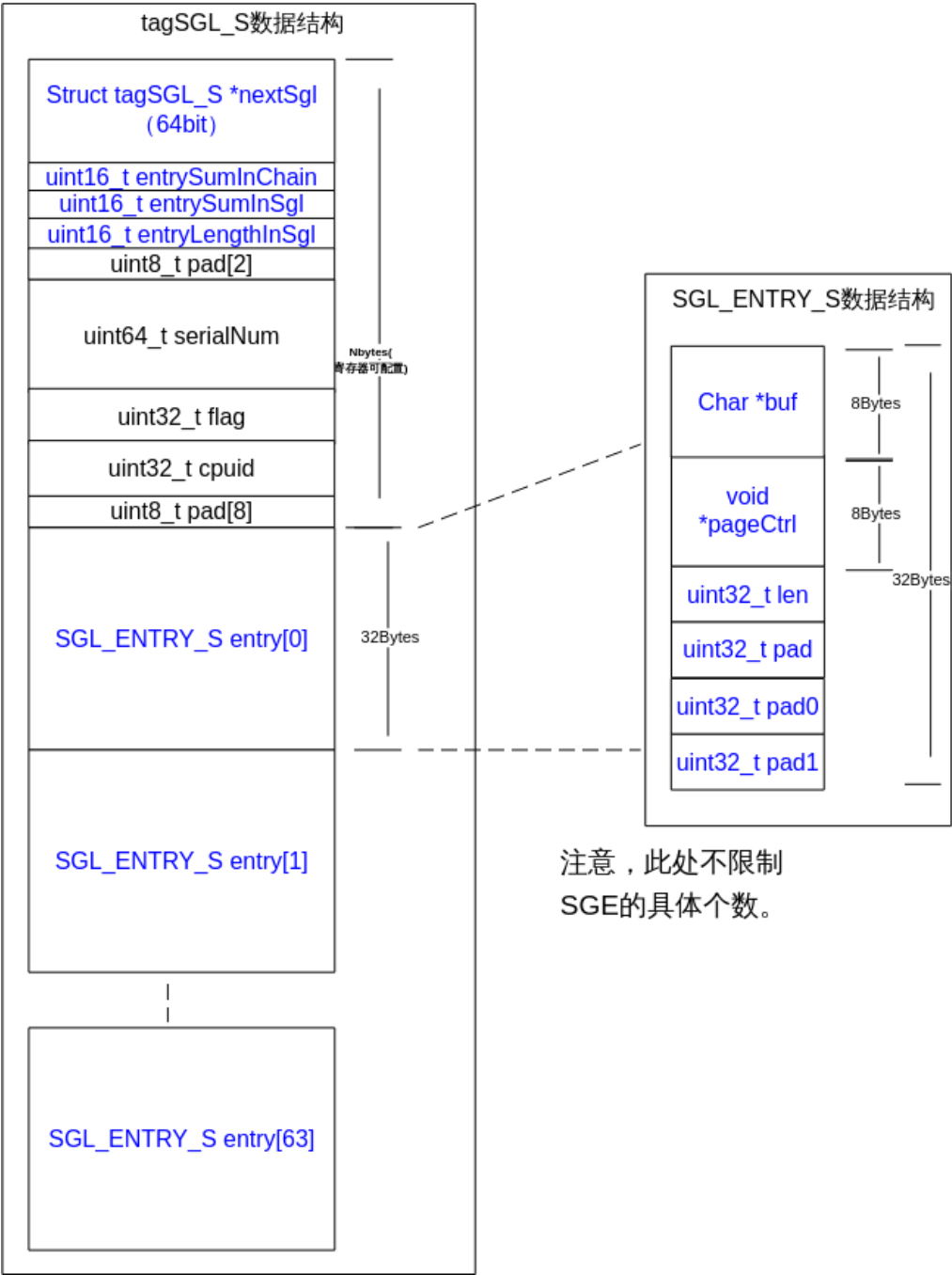
Pbuffer形式访问是指数据存放在指定的连续内存中，BD描述符中的Addr_Type =0b000。这种存放方式对于长度较短的数据比较适合，不需要采用SGL形式或者PPE List形式存放。

8.6.4.2 SGL 形式访问

SGL形式访问是指数据存放在SGL定义的数据格式存放在内存中，BD描述符中的Addr_Type =0b001。[图8-39](#)为SGL和SGE的数据格式，右边的单元为SGE（SGL_ENTRY_S）。SGE是业务软件申请数据内存的基本结构，本数据结构长度为32Byte，它包括了指向数据存储区的指针、长度（单位：Byte），页面控制头地址，PAD保留未用。左边的是SGL结构，包含了下一个SGL指针nextSgl、SGL链中的SGE总个数entrySumInChain、本SGL中包含的sgl_entry有效的个数entrySumInSgl、本SGL中sgl_entry的个数entryLengthInSgl、以及若干个SGE的聚散表结构。本SGL的首个SGE的起始地址偏移由寄存器配置，因此SGL结构中蓝色字体的域段是固定的，黑色字体的域段是软件可以灵活控制长度和内容的。多个SGL可组成一条链，由当前SGL的nextSgl指针指向下一个SGL的首地址。



图 8-39 SGL 和 SGE 的数据格式



须知

1. 蓝色字体域段为软硬件共用，其含义、大小和位置软件不能随意调整，其余字段软件可自行定义。
2. 每个SGL中的SGE个数为entryLengthInSgl，但是其中有效的SGE个数根据entrySumInSgl来定，至少要有1个SGE）。在多SGL链场景下，除第一个SGL外的其他SGL的entrySumInChain域即“SGL链中的SGE总个数”字段无效，允许随机配置。

8.6.4.3 PPE List 形式访问

PPE形式访问是指数据存放在PPE定义的数据格式存放在内存中，BD描述符中的Src_Addr_Type = 0b011。[图8-41](#)为PPE的数据格式，本数据结构长度为32Byte，它包括了指向数据存储区的指针、长度（单位：Byte），buff地址，buff地址偏移。[图8-40](#)为PPE数据流图，通过填写BD前8个word域段，指示下一个PPE指针nextAddr、PPE链中的RefItselfAddr、data_offset和datalength；

图 8-40 PPE 数据流示意图

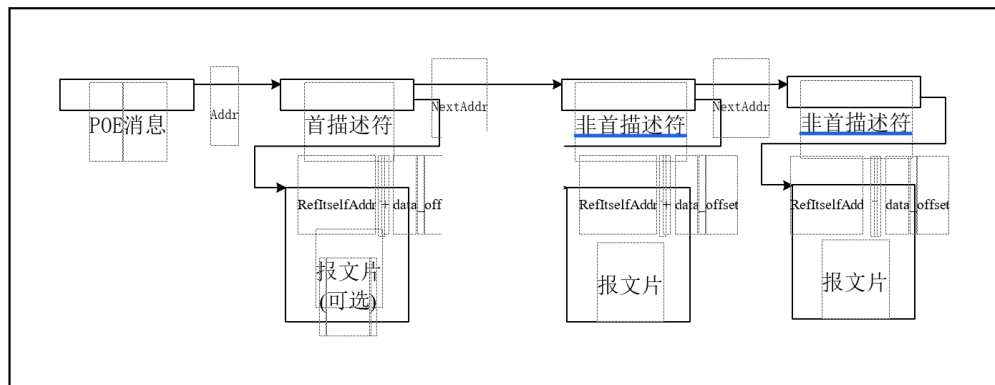
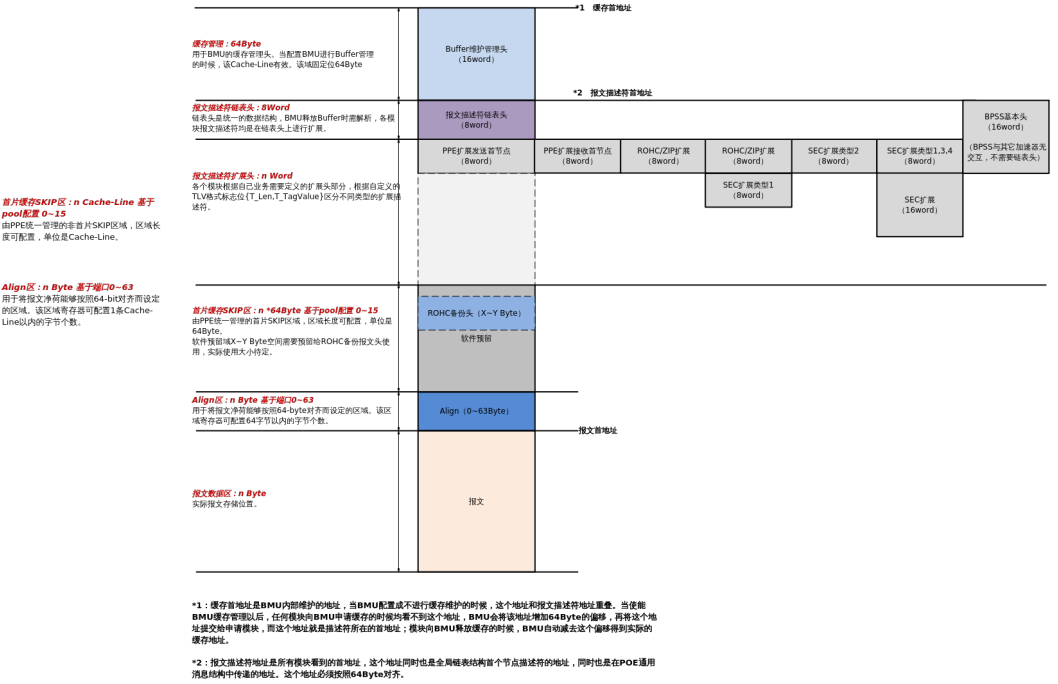




图 8-41 PPE List 单链示意图



须知

软件下载PPE链表任务时, 需要保证PPE 链表首节点 Data_length不为零。

8.6.5 软硬件接口

无

8.6.5.1 POE 接口工作方式

SEC的POE工作方式操作步骤如下:

- 步骤1** 软件将准备给SEC的任务添加到POE跟SEC绑定的Group里。
- 步骤2** POE通知SEC有任务待处理。
- 步骤3** SEC的任务管理逻辑向POE模块发起GET_POE请求, 获取任务。
- 步骤4** SEC的任务管理逻辑将任务调度给SAA, 并记录该任务的TAG, 用于Order保序。
- 步骤5** SAA按照SEC BD的指示对报文进行处理。
- 步骤6** SAA处理完报文后回写SEC BD。
- 步骤7** SAA将任务交回给任务管理逻辑, 任务管理逻辑根据之前记录的保序信息指示SAA能否将任务写入任务输出队列中。
- 步骤8** SEC的任务管理逻辑对任务输出队列中的任务依次发起ADD_PACKET操作, ADD到POE相应Group (在SEC BD中指示) 中。



步骤9 软件POE的SEC所ADD的Group里获取SEC处理过的任务进行后处理。

步骤10 POE使用Credit机制释放SEC中的ADD_PACKET资源。

----结束

8.6.5.2 长数据 HASH 操作流程

步骤1 软件把待处理的长HASH运算拆分成多个分片，长度要求如下：

每个分片最大长度为16MB~513B；

除最后一个分片外，其余所有分片长度都必须为64B整数倍。

步骤2 软件按照如下方式编辑BD，软件每次只能向对应队列添加一个BD，并且需要等待之前的BD已经处理完才能添加新的BD——如果超过250ms仍未等到之前的BD处理完成，软件报错：

配置首Hash部分时要求：BD配置AI_GEN域为0b0，Auth_Pad域为0b1，BD其余配置项和只做HASH的配置情况相同；

配置在做中间部分Hash时要求：BD配置AI_GEN域为0b1，Auth_Pad域为0b1，把上一个HASH的计算结果放在地址Auth_IVIN_Addr中，BD其余配置项和只做HASH的配置情况相同；

配置在做尾Hash部分时要求：BD配置AI_GEN域为0b1，Auth_Pad域为0b0，把上一个HASH的计算结果放在地址Auth_IVIN_Addr中，同时配置Long_Auth_Data_Len = 整个长数据Hash运算的报文总长度，总长度以bit为单位。

步骤3 待当前BD处理完成并回写到相应队列后，软件判定是否已经把全部分片做完，如果已经做完了，则长数据HASH运算结束，否则再回到**步骤2**执行。

----结束

8.6.5.3 BD FIFO（安全任务）接口工作方式

SEC的BD FIFO工作方式操作步骤如下：

步骤1 安全世界软件查询SEC_BDF_EN（0x0000），若bdf_push_en为0x0，需要配置为0x1，若bdf_pop_en为0x0，需要配置为0x1。

步骤2 安全世界软件查询SEC_BDF_FIFO_STATUS（0x0128），若bdf_fifo_full为0x0，则可以通过配置SEC_BDF_DATA_LOW（0x0100）和SEC_BDF_DATA_HIG（0x0104）（注意：必须保证SEC_BDF_DATA_LOW配置完成后再配置SEC_BDF_DATA_HIG，两者到达SEC slv口的顺序不允许颠倒）下发任务BD的地址。

步骤3 SEC根据BD FIFO获取的SEC描述符地址，获取SEC BD和各级链表描述符，并根据描述符的内容完成相关处理，然后完成BD的回写。

步骤4 任务完成方式查询：当（一段时间内）BD FIFO工作方式的任务都下发完成后，间隔若干时间（软件根据下发任务长度及数量自定义，至少要大于0.1μs），安全世界软件循环查询SEC_BDF_PKG_DONE_CNT（0x0110）和SEC_BDF_PKG_GET_CNT（0x010C），在两个寄存器没发生卷绕情况下，SEC_BDF_PKG_DONE_CNT加SEC_BDF_SPOP_CNT（0x031C）等于SEC_BDF_PKG_GET_CNT（0x010C），表示SEC已完成之前所有BD FIFO工作方式下发的任务的处理——如果超过2.5s仍未查询到两寄存器值相等，软件报错。



步骤5 安全世界软件查询回写的SEC BD，并确认SEC BD完成状态。

----结束

须知

如果一个SEC BD被识别为错误BD，那么无论DE域值如何配置都是回写到原地址。

8.6.5.4 BD FIFO 工作方式资源回收

因为BD FIFO工作方式不存在和POE和BMU的交互，所以BD FIFO资源回收主要体现在未完成的BD任务需要被安全CPU及时回收。安全CPU通过如下步骤完成资源回收：

- 步骤1** 配置SEC_BDF_EN寄存器，偏移地址为0x0000：配置为0x0，关闭BD FIFO工作方式的任务入口和出口。
- 步骤2** 查询SEC_BDF_FIFO_STATUS寄存器，偏移地址为0x128，当域段bdf_fifo_empty为0x0，表征SEC BD FIFO非空，请继续执行**步骤3**；若bdf_fifo_empty为0x1，表征SEC BD FIFO已经排空，请跳至**步骤5**。
- 步骤3** 配置SEC_BDF_SW_POP_EN为1，偏移地址为0x118：内部的BD FIFO控制逻辑检查到该寄存器为1时，把BD FIFO中的数据读出并锁存到寄存器。
- 步骤4** 查询SEC_BDF_SW_POP_DAT_LOW和SEC_BDF_SW_POP_DAT_HIG，偏移地址为0x120、0x124：获得BD FIFO弹出数据；若数据为0x5A5A5A5A，表明软件读BD FIFO操作和SEC访问BD FIFO出现冲突，此时BD FIFO优先响应SEC内部访问，因此返回0x5A5A5A5A；获取弹出数据后继续执行**步骤2**；但如果已连续1000次操作都是查询到0x5A5A5A5A，软件报错。
- 步骤5** 循环查询SEC_BDF_STATUS寄存器，偏移地址为0x0114：若寄存器值为0x0或0x3，则表明SAA中已经没有正在处理的BD FIFO任务，即已完成了本流程；若寄存器值为0x1或0x2，则表明SAA中可能有正在处理的BD FIFO任务，重复**步骤5**；如果超过250ms仍未查询到寄存器值为0x0或0x3，则BD FIFO工作方式资源回收失败。

须知

软件执行资源回收的时候需要注意避免导致BD FIFO出现空读现象，否则将导致SEC产生BD FIFO空读异常原始中断。

----结束

8.6.5.5 单模资源回收和丢弃流程

SEC资源回收（含丢弃）流程如下：

排空已获取任务：

- 步骤1** 关闭任务入口。配置VF_SEC_QUE_CFG寄存器（偏移地址0x063C）为0x2，即关闭get poe使能，但保留add poe使能。
- 步骤2** 间隔一定时间（ $\geq 1\mu s$ ）循环查询VF_SEC_QUE_CFG寄存器，直至vf_clr_done域变为1（VF_SEC_QUE_CFG寄存器，偏移地址0x063C被配置为0x2后，将不再发起新的GET



POE操作，等所有已取回的任务都ADD POE完毕后，将vf_clr_done拉高），即表明该VF的任务队列及SAA中任务都已排空；如果超过100ms仍未查询到vf_clr_done域变为1，则排空已获取任务的操作失败，软件记录此种失败。

----结束

8.6.5.6 单模内部模块复位流程

如果一个VF下的所有加速器都进行复位，称之为单模复位。

- 步骤1** 进行资源丢弃，具体流程参照“[单模资源回收和丢弃流程](#)”。
- 步骤2** 打开该业务统计寄存器的读清使能：配置VF_CNT_CLR_CE（偏移地址0x0610）为0x1。
- 步骤3** 清除该业务模式的所有统计寄存器值。
- 步骤4** 清除ACC_VF_ABNORMAL_INT_SOURCE所有源中断。
- 步骤5** 配置VF_FUNC_EN为0，关闭工作使能。

----结束

8.6.5.7 单模式初始化流程

单模式初始化流程是在其他业务模式正常工作的时候，开启一种新的业务模式执行包处理任务，或者某个模式单模复位后的初始化。单模式初始化要求不影响其他业务模式的正常工作。

VF寄存器组前置配置：

- 步骤1** 配置PF SR_IOV_VF_OFFSET寄存器，偏移地址0x012C：配置物理VF寄存器相对VF BAR0偏移的stride个数。
- 步骤2** 配置PF POE页签下 SR_IOV_CTL寄存器，偏移地址0x0108：在 vf_en_bitmap域配置该VF配置使能。

配置PF POE页签下SEC_VF_WEIGHT寄存器，偏移地址0x08C0：配置对应VF处理任务的权重。

配置PF POE页签下SEC_VF_POWER寄存器，偏移地址0x0900 0x0904：配置SEC每个VF能调度的最大包个数。
- 步骤3** VF计数器配置及中断配置。

配置VF POE_VF页签ACC_VF_ABNORMAL_INT_ENABLE寄存器，偏移地址0x0200：配置对应VF异常中断使能。

配置VF POE_VF页签VF_CNT_CLR_CE寄存器，偏移地址0x0610：配置对应VF读清寄存器读清使能。
- 步骤4** 配置poe使能和工作使能。

配置VF POE_VF页签 SEC_QUE_CFG寄存器，偏移地址0x063C：配置VF get POE使能。

配置VF POE_VF页签VF_FUNC_EN寄存器，偏移地址0x0608：配置VF工作使能。

----结束



8.6.6 BD FIFO 工作方式任务控制

BD FIFO寄存器SEC_BDF_EN（偏移地址为0x0000）提供了BD FIFO任务入口和出口控制，其中bdf_push_en控制了BD FIFO的任务入口，bdf_pop_en控制了BD FIFO的任务出口，详细配搭方式如表8-12所示。

表 8-12 BD FIFO 任务控制组合

bdf_pop_en	bdf_push_en	组合结果
0	0	关闭BD FIFO的任务出口和入口： 1. 禁止安全CPU向BD FIFO下发任务。 2. 不允许SAA获取新的BD FIFO任务。 3. SAA可以返回已经完成的BD FIFO任务。 典型应用场景：SEC安全复位，BD FIFO任务回收。
0	1	仅关闭BD FIFO任务出口： 1. 允许安全CPU向非满的BD FIFO下发任务。 2. 不允许SAA获取新的BD FIFO任务。 3. SAA可以返回已经完成的BD FIFO任务。 典型应用场景：NA。
1	0	仅关闭BD FIFO任务入口： 1. 禁止安全CPU向BD FIFO下发任务。 2. 允许SAA获取新的BD FIFO任务。 3. SAA可以返回已经完成的BD FIFO任务。 典型应用场景：任务BD FIFO任务排空。
1	1	打开BD FIFO的任务出口和入口： 1. 允许安全CPU向BD FIFO下发任务。 2. 允许SAA获取新的BD FIFO任务。 3. SAA可以返回已经完成的BD FIFO任务。 典型应用场景：安全运行。

交互数据结构请参见 SEC BD（SQE）。

CRC计算方法如下：

```
//-----  
-----  
//1、配表数据计算CRC方法  
//1 ) 配表数据进行32B对齐，padding数据填0x0。  
//2 ) seed设置为0xffffffff，调用sa_crc32(unsigned int seed ,const unsigned char *buf, unsigned  
int len)进行计算；有多个2KB时，再次调用sa_crc32()时，seed值为上次CRC的32bit返回值按位取  
反。  
//-----  
-----  
/*  
* Table for the AUTODIN/HDLC/802.x CRC.
```



```
*
* Polynomial is
*
*  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 +$ 
*    $x^5 + x^4 + x^2 + x + 1$ 
*/
const uint32_t crc32table_le[256] =
{
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419,
0x706af48f, 0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4,
0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07,
0x90bf1d91, 0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856,
0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4,
0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3,
0x45df5c75, 0xdcd60dcf, 0xabd13d59, 0x26d930ac, 0x51de003a,
0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599,
0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190,
0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f,
0x9fbfe4a5, 0xe8b8d433, 0x7807c9a2, 0xf00f934, 0x9609a88e,
0xe10e9818, 0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
0xb66b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed,
0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbbeb8e, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3,
0xfbd44c65, 0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a,
0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5,
0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa, 0xbe0b1010,
0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17,
0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6,
0x03b6e20c, 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615,
0x73dc1683, 0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
0xe0ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1, 0xf00f9344,
0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf62575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a,
0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1,
0xa6bc5767, 0x3fb506dd, 0x48b2364b, 0xd80d2bda, 0xaf0a1b4c,
0x36034af6, 0xa1047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef,
0x46699e79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe,
0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31,
0x2cd99e8b, 0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c,
0x026d930a, 0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb1b38, 0x92d28e9b,
0xe5d5be0d, 0x7cdcefb7, 0x0bdbfd21, 0x86d3d2d4, 0xf1d4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1,
0x18b74777, 0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45, 0xa00ae278,
0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7,
0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66,
0x37d83bf0, 0xa9bcae53, 0xdeb9ec5, 0x47b2cf7f, 0x30b5ffe9,
0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605,
0xcdd70693, 0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8,
0x5d681b02, 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b,
0x2d02ef8d
};
uint32_t sa_crc32(unsigned int seed, const unsigned char *buf, unsigned int len)
```




```
{
    unsigned int crc32 = seed;
    unsigned int i = 0;
    for ( i = 0; i < len; i++)
    {
        crc32 = crc32table_le[((crc32 & 0xff) ^ buf[i]) & 0xff] ^ (crc32 >> 8);
    }
    return (~crc32);
}
//-----
//2、报文payload计算CRC方法
//1) 对下发给SA的完整TCP报文数据不要padding数据，seed设置为0x0，直接调用
sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。最后把返回值赋值给
BD的pkt_crc16域段。
//2) 把返回值赋值给BD的pkt_crc16域段。
//-----
/* Polynomial num is 1021
const uint16_t crc16table_le[256] =
{
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcdbc, 0xfbfb, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedeae, 0xfdf8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};
#define ALIGN_SIZE 32
uint16_t sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)
{
    uint16_t crc16 = seed;
    unsigned int i = 0;
    unsigned int alg_len = 0;
    alg_len = (len / ALIGN_SIZE) * ALIGN_SIZE + (((len % ALIGN_SIZE) == 0) ? 0 : ALIGN_SIZE);
    for ( i = 0; i < alg_len; i++)
```



```
{
if (i < len) {
crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ buf[i] & 0xff) ^ (crc16 << 8);
} else {
crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ 0x0) & 0xff] ^ (crc16 << 8);
}
}
return crc16;
}
//-----
//3、BD计算CRC方法
//1) 对下发给SA的64byte的BD截取前面48byte的数据，seed设置为0x0，直接调用
sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。最后把返回值赋值给BD的
pkt_crc16域段。
//2) 把返回值的高8bit赋值给BD的pkt_crc16域段的低8bit，把返回值的低8bit赋值给BD的pkt_crc16
域段的高8bit。
//-----
uint16_t sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)
{
uint16_t crc16 = seed;
unsigned int i = 0;
for ( i = 0; i < len; i++)
{
crc16 = crc16table_le[(((crc16 >> 8) & 0xff) ^ buf[i] & 0xff) ^ (crc16 << 8);
}
return crc16;
}
```

1. 配表数据计算CRC方法

- 配表数据进行32B对齐，padding数据填0x0。
- seed设置为0xFFFFFFFF，调用sa_crc32(unsigned int seed, const unsigned char *buf, unsigned int len)进行计算；有多个2KB时，再次调用sa_crc32()时，seed值为上次CRC的32bit返回值按位取反。

2. 报文payload计算CRC方法

- 对下发给SA的完整TCP报文数据不要padding数据，seed设置为0x0，直接调用sa_crc16_align32(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。
- 把返回值赋值给BD的pkt_crc16域段。

3. BD计算CRC方法

- 对下发给SA的64byte的BD截取前面46Byte的数据，seed设置为0x0，直接调用sa_crc16(uint16_t seed, const uint8_t *buf, uint32_t len)进行计算。
- 把返回值的高8bit赋值给BD的bd_crc16域段的低8bit，把返回值的低8bit赋值给BD的bd_crc16域段的高8bit。
- 为了将部分硬件匹配引擎处理错误的信息送给软件，通过ADD_PACKET数据结构的User_Data(word5中)上报这些信息：
- User_Data[3:0]：为0x0，表示BD执行正确；bit[0]为1，表示硬件引擎收到的任务存在错误；bit[1]为1，表示该匹配任务超时，只对同步任务生效；bit[2]为1，表示POE所携带的地址非法。

8.6.7 性能和功耗配置



8.6.7.1 时钟门控

SEC提供动态时钟门控功能，此功能的开关为SEC_CONTROL寄存器的clk_gate_enable域：

当使能动态时钟门控后，SEC会自动关断没有任务处理的SAA的时钟，当有任务待某SAA处理时又会自动打开其时钟；

SEC动态关断某SAA时钟的时候，不会关断SAA外围的IVIN、KEY、SGE、AW等模块时钟；不过当某个CLUSTER中所有SAA都没有任务时，SEC会关掉整个SEC_CLUSTER_TOP的时钟。

8.6.8 固件编程接口

步骤1 申请一段至少256B大小的地址空间仅供加速器读使用。

步骤2 将申请地址的高32bit配置到寄存器SEC_CHAIN_ABN_RD_ADDR_HIG低32bit配置到寄存器SEC_CHAIN_ABN_RD_ADDR_LOW中。

步骤3 将申请到的空间大小（至少256B）配置到寄存器SEC_CHAIN_ABN_RD_LEN中。

步骤4 申请一段至少256B大小的地址空间仅供加速器写使用。

步骤5 将申请地址的高32bit配置到寄存器SEC_CHAIN_ABN_WR_ADDR_HIG，低32bit配置到寄存器SEC_CHAIN_ABN_WR_ADDR_LOW中。

步骤6 将申请到的空间大小（至少256B）配置到寄存器SEC_CHAIN_ABN_WR_LEN中。

步骤7 根据空间属性，配置SEC寄存器SEC_CHAIN_ABN_AW_CACHE_CFG、SEC_CHAIN_ABN_AR_CACHE_CFG、SEC_CHAIN_ABN_AW_USER_CTRL0、SEC_CHAIN_ABN_AW_USER_CTRL1、SEC_CHAIN_ABN_AW_USER_CTRL2、SEC_CHAIN_ABN_AR_USER_CTRL0、SEC_CHAIN_ABN_AR_USER_CTRL1、SEC_CHAIN_ABN_AR_USER_CTRL2。若申请的是物理地址，则上述寄存器保持为默认值，若申请的是虚拟地址，则需配置上述寄存器，SEC_CHAIN_ABN_AR_USER_CTRL1、SEC_CHAIN_ABN_WR_USER_CTRL1[14:12]配置为0b001。

须知

建议申请物理空间用于加速器数据排空。

----结束

8.6.9 AEAD 算法描述符配置

8.6.9.1 AES-CCM 算法描述符配置

SEC支持基于AES的AES_CCM算法，即Counter with CBC_MAC算法，用于IPSEC，符合RFC4309协议标准；

AES CCM算法分加密和解密两种方向。在CCM加密方向，SEC先认证（AES-CBC）后加密（AES-CTR）。在CCM解密方向，SEC先解密（AES-CTR）再解认证（AES-CBC）。



按照算法协议，完成CCM算法需要如下参数：

- 1. AES算法密钥，AES-CCM算法加解密和认证算法共用一套密钥，即软件只需要配置加解密密钥Cipher_Key_Addr。
- 2. AES CCM算法待处理报文。
- 3. AAD数据, 参与认证运算。
- 4. CBC IVIN，执行AES-CCM算法时使用的认证IVIN。
- 5. CTR IVIN，执行AES-CCM算法时使用的加解密IVIN。

Auth_Len	认证报文长度；当执行AES-CCM和AES-GCM算法时，表示AAD数据的长度；单位：Byte。
----------	--

AES-CCM算法的加解密报文，长度地址解析方式和其他主控加解密算法一致。

但需要注意的是必须保证：Auth_Source_Offset + Auth_Length <= Cipher_Source_Offset。

表 8-13 AES-CCM 算法参数配置

-	C_Alg (3bit)	C_Mode (3bit)	Cipher (2bit)	Auth (2bit)
AES-CCM加密	AES 010	CCM 101	加密 01	算出的MAC值写到外部DDR / Cache中 00
AES-CCM解密	AES 010	CCM 101	解密 10	算出的MAC值与指定地址的MAC值进行校验 00

须知

注意的是必须保证：Auth_Source_Offset + Auth_Length <= Cipher_Source_Offset。

表 8-14 AES-CCM 算法参数配置

用户态 算法	可设置key 长(b)	可设置IV 长(b)	认证头长 度(B)	认证长度(B)	包长限 制(B)	包长异 常检测
ccm(aes)	128、 192、256	128	0~65279	4,6,8,10,12,14,16	字节对 齐	硬件
ccm(sm 4)	128	128	0~65279	4,6,8,10,12,14,16	字节对 齐	硬件



8.6.9.2 AES-GCM 算法描述符配置

SEC支持基于AES的AES_GCM算法，即Counter with GMAC算法，符合RFC4106协议标准。

AES-GCM算法分加密和解密两种方向。在GCM加密方向，SEC执行加密算法并且输出MAC值。在GCM解密方向，SEC执行解密算法并且完成对MAC值的校检。

AES-GCM算法同时执行加解密和认证运算。

依据RFC4106协议，IP SEC业务场景下，AES-GCM算法仅允许IVIN初始向量长度配置为12个Byte。

依据RFC4106协议，IP SEC业务场景下，AES-GCM算法仅允许AAD报文长度配置为8个Byte或者12个Byte，加解密报文长度不限。

Auth_Len	认证报文长度；当执行AES-CCM和AES-GCM算法时，表示AAD数据的长度；单位：Byte。
----------	--

AES-GCM算法的加解密报文，长度地址解析方式和其他主控加解密算法一致。

但需要注意的是必须保证：Auth_Source_Offset+Auth_Length<=Cipher_Source_Offset。

依据RFC4106协议，IP SEC业务场景下，AES-GCM算法的ICV长度（T_Length）仅允许配置为16Byte（推荐），12Byte或者8Byte。SEC数据结构接受8Byte~16Byte之间的任何配置。

表 8-15 AES-GCM 算法参数配置表

-	C_Alg (3bit)	C_Mode (3bit)	Cipher (2bit)	Auth(2bit)
AES-GCM加密	AES 010	GCM 110	加密 01	算出的MAC值写到外部DDR / Cache中 00
AES-GCM解密	AES 010	GCM 110	解密 10	算出的MAC值与指定地址的MAC值进行校验 00

须知

使用AES CCM/GCM 流模式时，软件需要关注MAC地址写出开关，配置SEC_PF_REG页签的SEC_ECO_RW（偏移0x0580），配置为0x700。

8.7 初始化

8.7.1 软复位

软件需要按照下述流程对SEC进行软复位准备，各步骤都要顺序执行。



步骤1 软件确保SEC中BDFIFO接口无剩余任务：

等待SEC_BD_FIFO页签的SEC_BDF_PKG_DONE_CNT (0x108) 与 SEC_BDF_SPOP_CNT (0x031C) 之和等于SEC_BDF_PKG_GET_CNT (0x010C) 。

步骤2 软件确保SEC中POE接口无剩余任务：

等待SEC_POE_VF_REG页签的SEC_QUE_CFG (0x063C) 的vf_clr_done寄存器值为1。

步骤3 关闭总线接口：

1. 下发关闭总线接口请求：

把《 Hi309a 寄存器手册 》的偏移为0x0的bit[0] (域ctrl_shutdown_req) 配置为0x1，则不再发起新的总线命令。

2. 查询总线命令处理状态：

查询《 Hi309a 寄存器手册 》的寄存器AM_CURR_TRANS_RETURN (偏移为0x150) 为0x3时，表征之前发出的所有总线操作都已经被Master接口处理完毕。

3. 如果软件在自己规定时间内 (建议值不超过10ms) 未查询到上述寄存器值变为0x3，这预示出现总线死锁，复位准备无法完成，建议记录各模块总线相关信息后复位整个芯片。

须知

发生此种情况预示出现总线死锁，推荐做法是记录各模块总线相关信息后复位整个芯片。

步骤4 软复位复位SEC：

当SEC的复位准备完成后，通过操作《 HI309a PERI Subctrl 寄存器描述 》中有关SEC的寄存器，来完成SEC的复位操作：

1. 配置SC_SEC_CLK_DIS为0x2，关闭模块时钟。
2. 查询SC_SEC_CLK_ST，若查询值为0x1，则表示模块时钟关闭。
3. 配置SC_SEC_RESET_REQ为0x2，对SEC模块进行复位。
4. 查询SC_SEC_RESET_ST，若查询值为0x2，则表示模块处于复位状态。
5. 配置SC_SEC_CLK_EN为0x2，打开模块时钟。
6. 查询SC_SEC_CLK_ST，若查询值为0x3，则表示模块时钟打开。
7. 等待500ns，配置SC_SEC_CLK_DIS为0x2，关闭模块时钟。
8. 查询SC_SEC_CLK_ST，若查询值为0x1，则表示模块时钟关闭。
9. 软件判断软复位已经完成，执行以下步骤进行解复位。
10. 配置SC_SEC_RESET_DREQ为0x2，对SEC模块进行解复位。
11. 查询SC_SEC_RESET_ST，若查询值为0x8，则表示模块处于解复位状态。
12. 配置SC_SEC_CLK_EN为0x2，打开模块时钟。
13. 查询SC_SEC_CLK_ST，若查询值为0x3，则表示模块时钟打开。

步骤5 软件读清SEC寄存器状态：

SEC计数类寄存器没有复位，需要软件需要执行以下操作完成SEC模块相关状态和统计寄存器清空功能。



1. 配置《Hi309a SEC 寄存器描述》中SEC_PF_REG的SEC_CNT_CLR_CE.
sec_cnt_clr_ce为1，读清操作使能。
2. 软件执行读操作清空SEC_PF_REG页签中所有CNT_CYC和CNT类型寄存器。
3. 配置《Hi309a SEC 寄存器描述》中SEC_PF_REG的SEC_CNT_CLR_CE.
sec_cnt_clr_ce为0，读清操作去使能。

----结束

8.7.2 初始化

8.7.2.1 安全启动配置

解复位SEC:

当SEC的上电完成后，通过操作《Hi309a 寄存器手册》中有关SEC的寄存器，来完成SEC的解复位操作：

- 步骤1** 配置SC_SEC_CLK_DIS为0x3，关闭模块时钟。
- 步骤2** 查询SC_SEC_CLK_ST，若查询值为0x000000，则表示模块时钟关闭。
- 步骤3** 配置SC_SEC_RESET_REQ为0xB，对SEC模块进行复位。
- 步骤4** 查询SC_SEC_RESET_ST，若查询值为0xB，则表示模块处于复位状态。
- 步骤5** 配置SC_SEC_CLK_EN为0x3，打开模块时钟。
- 步骤6** 查询SC_SEC_CLK_ST，若查询值为0x3，则表示模块时钟打开。
- 步骤7** 等待500ns，配置SC_SEC_CLK_DIS为0x3，关闭模块时钟。
- 步骤8** 查询SC_SEC_CLK_ST，若查询值为0x0，则表示模块时钟关闭。
- 步骤9** 软件判断软复位已经完成，执行以下步骤进行解复位。
- 步骤10** 配置SC_SEC_RESET_DREQ为0xB，对SEC模块进行解复位。
- 步骤11** 查询SC_SEC_RESET_ST，若查询值为0x0，则表示模块处于解复位状态。
- 步骤12** 配置SC_SEC_CLK_EN为0x3，打开模块时钟。
- 步骤13** 查询SC_SEC_CLK_ST，若查询值为0x3，则表示模块时钟打开。

----结束

须知

当发现解复位后相应寄存器未能使能，需要关注《Hi309a SysCtrl 寄存器》去软复位等信号。

8.7.2.2 公共初始化

- 步骤1** 配置SEC_PF_REG页签的SEC_BD_PACKET_OST_CFG（偏移0x020C）SEC读取BD时的Outstanding。



- 步骤2** 软件配置SEC_PF_REG页签的SEC_CONTROL（偏移0x0200）SEC全局配置。
- 步骤3** 软件配置SEC_PF_REG页签的心跳阈值SEC_FSM_MAX_CNT（偏移0x0124）。
- 步骤4** 软件配置SEC_CORE_REG页签的SEC_SGL_OFFSET_CONTROL（偏移0x0108）的SGL数据结构偏移，配置SEC_CORE_REG页签的SEC_DIF_CRC_INIT（偏移0x0114）的CRC初始值。
- 步骤5** 软件配置SEC_PF_REG页签的SEC_MEM_START_INIT（偏移0x0100）的MEM自动初始化。
- 步骤6** 软件查询SEC_PF_REG页签的SEC_MEM_INIT_DONE（偏移0x0104）的mem_init_done是否为1（初始化完成），如果初始化完成进入**步骤7**。
- 步骤7** 软件根据需要配置其他阈值，中断使能，DFX相关的寄存器等。
- 步骤8** 软件根据需要配置SEC_PF_REG页签的SEC_CNT_CLR_CE（偏移0x0120）的PF统计寄存器读清使能。
- 步骤9** 软件根据系统需求配置SEC_PF_REG页签的SEC_SAA_EN（偏移0x0214），配置使用哪些SAA。
- 步骤10** 软件根据系统需求配置SEC_CORE_REG页签的SEC_AXI_CACHE_CFG（偏移0x0210）、SEC_SNPATTR_CFG（偏移0x0218）的QOS属性，SEC_INTERFACE_USER_CTRL0（偏移0x0220）、SEC_INTERFACE_USER_CTRL1（偏移0x0224）的axusr相关，无特殊需求可不配置。
- 步骤11** 链表寄存器模式配置：
配置SEC_LINK_RSV寄存器，偏移地址0x0910：配置为0x10004。
- 步骤12** 软件配置SEC_PF_REG页签的SEC_ECO_RW（偏移0x0580），配置为0x700。
- 步骤13** 根据应用需求，POE接口、BDFIFO接口和秘钥加载的初始化。
- 结束

8.7.2.3 BD FIFO 接口初始化

只要对SEC实行了模块复位，SEC模块属性初始化完成后，存在如下场景时，需要对SEC BD FIFO进行初始化：

芯片安全运行，例如：需要通过SEC调用芯片HUK功能。

- 步骤1** 配置BD FIFO模式启用标识。

配置BDF_EN_STATUS寄存器，偏移地址0x0004：配置为1，标明启用BD FIFO模式。

- 步骤2** BD FIFO属性寄存器配置。

1. 配置SEC_BDFIFO_REG页签ACC_STREAMID_S寄存器，偏移地址：配置BD FIFO模式下的AXI AWUSER/ARUSER Stream ID。
2. 配置SEC_BDF_CFG_AWUSER寄存器，偏移地址：配置BD FIFO模式下SEC各种数据的AXI AWUSER其他关键域。
3. 配置SEC_BDF_CFG_ARUSER寄存器，偏移地址：配置BD FIFO模式下SEC各种数据的AXI ARUSER其他关键域。
4. 配置SEC_BDF_CFG_PD_TAG寄存器，偏移地址：配置BD FIFO模式队列TAG值。
5. 配置SEC_BDF_CNT_CLR_CE寄存器，偏移地址：配置BD FIFO模式寄存器读清使能。



步骤3 BD FIFO中断寄存器配置。

1. 配置寄存器，偏移地址：配置BD FIFO模式中断使能寄存器。
2. 配置寄存器，偏移地址，写清ECC_2BIT_ERR_RINT寄存器。

步骤4 清除统计寄存器。

1. 配置PF_REG页签寄存器，偏移地址：读清SEC_ECC_2BIT_CNT寄存器。
2. 配置SEC_BDFIFO_REG页签，偏移地址，读清SEC_BDF_PKG_DOING_CNT寄存器。
3. 配置SEC_BDFIFO_REG页签，偏移地址，读清SEC_BDF_PKG_GET_CNT寄存器。
4. 配置SEC_BDFIFO_REG页签，偏移地址，读清SEC_BDF_PKG_DONE_CNT寄存器。

步骤5 关闭读清使能寄存器。

配置SEC_BDFIFO_REG页签，偏移地址，配置SEC_BDF_CNT_CLR_CE寄存器不使能。

步骤6 BD FIFO任务开关配置。

配置SEC_BDF_EN寄存器，偏移地址：配置BD FIFO模式使能，打开BD FIFO POP和PUSH使能。

步骤7 链表寄存器模式配置。

配置SEC_LINK_RSV寄存器，偏移地址：配置为0x10004。

----结束

8.7.2.4 POE 接口初始化

步骤1 SEC内部属性配置。

配置SEC_POE_REG页签SEC_POE_PACKET_OST_CFG寄存器（默认值为0x10），偏移地址0x0880：配置访问POE的Outstanding深度。

步骤2 AXUSER的FA、FNA、RINVLD、PH、PKG配置（BD FIFO和POE工作方式共用）。

步骤3 SMMU bypass时的配置。

配置SEC_POE_REG页签的SEC_POE_AXI_USER_CFG寄存器，偏移地址0x0808，各cmd_type都需要配置为0。

步骤4 SMMU使能时的配置。

配置SEC_POE_REG页签的SEC_POE_AXI_USER_CFG寄存器，偏移地址0x0808，酌情配置ssv。

步骤5 SEC中断、计数器门限相关配置。

1. 配置SEC_POE_VF_REG页签ACC_VF_ABNORMAL_INT_ENABLE寄存器，偏移地址0x0200：配置异常中断使能。
2. 配置SEC_POE_REG页签SEC_CNT_CLR_CE寄存器，偏移地址0x0830：配置公共计数器读清寄存器读清使能。

步骤6 SEC访问外部模块相关配置。

1. 配置SEC_POE_REG页签ACC_NODE_CODE寄存器，偏移地址0x0678：配置访问POE的SourcePort。



2. 配置SEC_POE_REG页签ACC_VF_STREAMID寄存器，偏移地址0x0680：配置各VF访问总线时携带的streamid。

步骤7 SEC访问BD地址相对PD中BD地址的偏移配置SEC_POE_REG页签SEC_POE_BD_OFFSET寄存器，偏移地址0x0608。

步骤8 SEC次数和时间门限相关配置。

配置SEC_POE_REG页签SEC_GET_FAIL_THLD寄存器，偏移地址0x0828：配置GET POE连续失败次数门限寄存器。

----结束

8.7.2.5 SEC POE 工作方式下的 VF 业务相关初始化

SEC复位后初始化，需先配置PF寄存器的公共配置，公共配置完成后配置VF的业务配置，最后打开访问POE使能和工作使能。

-----PF寄存器组配置（地址范围
0x00_1000~0x00_1FFF）-----

步骤1 SEC访问外部模块相关配置。

1. 配置POE REG页签 ACC_VF_STREAMID寄存器，偏移地址0x0680~0x694：配置各VF访问总线时携带的streamid。
2. 配置POE REG页签 ACC_VF_VIS_POE_STRIDE寄存器，偏移地址0x06B8：配置POE相邻VF地址空间大小。
3. 配置POE REG页签 ACC_VF_ADD_POE_ADDR寄存器，偏移地址0x060C 0x0610：配置VF ADD POE时的地址。
4. 配置POE REG页签ACC_VF_VIS_POE_OFFSET寄存器，偏移地址0x06BC：配置每个VF访问POE的地址相对于POE访问VF基地址偏移的stride个数。
5. 配置POE REG页签ACC_VF_GET_POE_ADDR寄存器，偏移地址0x06D0 0x06D4：配置VF GET POE时的地址。
6. 配置POE VF REG页签VF_PD_BD_ERR_VQI，偏移地址0x0660，配置PD、BD错误的任务在ADD POE时使用的Vqi域。

步骤2 SEC 次数和时间门限相关配置。

1. 配置POE REG页签PF SEC_GET_FAIL_THLD寄存器，偏移地址0x0828：配置GET POE连续失败次数门限寄存器。
2. 配置PF REG页签 SEC_REQ_TRNG_TIME_TH寄存器，偏移地址0x012C：配置SEC申请TRNG的时间门限。

步骤3 VF寄存器组前置配置。

步骤4 SEC有5组物理业务寄存器，VF0~4寄存器组支持VF虚拟地址映射，最大支持5组VF；BDFIFO选用一组业务寄存器。

步骤5 配置POE REG页签 SR_IOV_VF_OFFSET寄存器，偏移地址0x012C：配置物理VF寄存器相对VF BAR0的stride个数。

步骤6 配置POE REG页签 SR_IOV_CTL寄存器，偏移地址0x0108：在 vf_en_bitmap域配置该VF配置使能。

1. 配置POE REG页签 SEC_VF_WEIGHT寄存器，偏移地址0x08C0：配置对应VF处理任务的权重。



2. 配置POE REG页签 SEC_VF_POWER寄存器，偏移地址0x0900 0x0904：配置SEC每个VF能调度的最大包个数。

-----VF/PF业务寄存器组配置-----

-----（地址范围VF0~VF4支持虚拟映射）-----

步骤7 VF中断配置及计数器配置。

1. 配置POE VF REG页签ACC_VF_ABNORMAL_INT_ENABLE寄存器，偏移地址0x0200：配置对应VF异常中断使能。
2. 配置POE VF REG页签VF_CNT_CLR_CE寄存器，偏移地址0x0610：配置对应VF读清寄存器读清使能。

步骤8 配置访问POE使能和工作使能。

1. 配置POE VF REG页签SEC_QUE_CFG寄存器，偏移地址0x063C：配置VF get POE和add POE使能。
2. 配置POE VF REG页签VF_FUNC_EN寄存器，偏移地址0x0608：配置VF工作使能。

----结束

8.7.2.6 KM 秘钥加载

SEC自动加载KM中所有秘钥的具体配置流程如下：（单独申请一种KEY可以直接从Step3开始执行）

步骤1 配置SEC_BD_FIFO_REG页签的KM_AUTO_REQ_START寄存器，偏移地址0x0040：写km_key_auto_req_cmd和km_key_auto_req寄存器。

步骤2 查询SEC_BD_FIFO_REG页签的KM_AUTO_REQ_DONE寄存器，偏移地址0x0044；如果km_key_auto_done为0，且km_key_auto_doing为1，则等待1μs后继续查询；如果km_key_auto_done为1，且km_key_auto_fail为1，则配置SEC_KM_KEY_INFO_CMD寄存器，偏移地址0x0324，查询SEC_KM_KEY_INFO寄存器，偏移地址0x0328，跳到下一步；如果km_key_auto_done为1，且km_key_auto_fail为0，则表示自动加载成功完成。

步骤3 配置SEC_BD_FIFO_REG页签的KM_REQ_START寄存器，偏移地址0x0038：写km_key_req_cmd和km_key_req寄存器。

步骤4 查询SEC_BD_FIFO_REG页签的KM_REQ_DONE寄存器，偏移地址0x003C；如果km_key_done为0，且km_key_doing为1，则等待1μs后继续查询；如果km_key_done为1，且km_key_fail为1，则配置SEC_KM_KEY_INFO_CMD寄存器，偏移地址0x0324，查询SEC_KM_KEY_INFO寄存器，偏移地址0x0328；如果km_key_done为1，且km_key_fail为0，则表示加载成功完成。

步骤5 下发BD前等待1μs。

----结束

8.7.3 软复位

软件需要按照下述流程对SEC进行软复位准备，各步骤都要顺序执行。

步骤1 软件确保SEC中BDFIFO接口无剩余任务。



1. 等待SEC_BD_FIFO页签的SEC_BDF_PKG_DONE_CNT (0x108) 与 SEC_BDF_SPOP_CNT (0x031C) 之和等于SEC_BDF_PKG_GET_CNT (0x010C)。

步骤2 软件确保SEC中POE接口无剩余任务。

1. 等待SEC_POE_VF_REG页签的SEC_QUE_CFG (0x063C) 的vf_clr_done寄存器值为1。

步骤3 关闭总线接口。

1. 下发关闭总线接口请求：
把《Hi309a 寄存器手册》的偏移为0x0的bit[0] (域ctrl_shutdown_req) 配置为0x1, 则不再发起新的总线命令。
2. 查询总线命令处理状态：
查询《Hi309a 寄存器手册》的寄存器AM_CURR_TRANS_RETURN (偏移为0x150) 为0x3时, 表征之前发出的所有总线操作都已经被Master接口处理完毕。
3. 如果软件在自己规定时间内 (建议值不超过10ms) 未查询到上述寄存器值变为0x3, 这预示出现总线死锁, 复位准备无法完成, 建议记录各模块总线相关信息后复位整个芯片。

须知

发生此种情况预示出现总线死锁, 推荐做法是记录各模块总线相关信息后复位整个芯片。

步骤4 软复位SEC。

当SEC的复位准备完成后, 通过操作《Hi309a PERI Subctrl 寄存器描述》中有关SEC的寄存器, 来完成SEC的复位操作:

1. 配置SC_SEC_CLK_DIS为0x2, 关闭模块时钟。
2. 查询SC_SEC_CLK_ST, 若查询值为0x1, 则表示模块时钟关闭。
3. 配置SC_SEC_RESET_REQ为0x2, 对SEC模块进行复位。
4. 查询SC_SEC_RESET_ST, 若查询值为0x2, 则表示模块处于复位状态。
5. 配置SC_SEC_CLK_EN为0x2, 打开模块时钟。
6. 查询SC_SEC_CLK_ST, 若查询值为0x3, 则表示模块时钟打开。
7. 等待500ns, 配置SC_SEC_CLK_DIS为0x2, 关闭模块时钟。
8. 查询SC_SEC_CLK_ST, 若查询值为0x1, 则表示模块时钟关闭。
9. 软件判断软复位已经完成, 执行以下步骤进行解复位。
10. 配置SC_SEC_RESET_DREQ为0x2, 对SEC模块进行解复位。
11. 查询SC_SEC_RESET_ST, 若查询值为0x8, 则表示模块处于解复位状态。
12. 配置SC_SEC_CLK_EN为0x2, 打开模块时钟。
13. 查询SC_SEC_CLK_ST, 若查询值为0x3, 则表示模块时钟打开。

步骤5 软件读清SEC寄存器状态:

SEC计数类寄存器没有复位, 需要软件需要执行以下操作完成SEC模块相关状态和统计寄存器清空功能。



- 1. 配置《Hi309a SEC 寄存器描述》中SEC_PF_REG的SEC_CNT_CLR_CE.
sec_cnt_clr_ce为1，读清操作使能。
- 2. 软件执行读操作清空SEC_PF_REG页签中所有CNT_CYC和CNT类型寄存器。
- 3. 配置《Hi309a SEC 寄存器描述》中SEC_PF_REG的SEC_CNT_CLR_CE.
sec_cnt_clr_ce为0，读清操作去使能。

----结束

8.8 DFX 应用指导

8.8.1 SEC 常见异常处理

常见的SEC BD配置错误、链表错误、总线错误、ECC错误等的处理方式，分类列如表8-16所示。

表 8-16 SEC 常见异常处理

bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TY PE/ WARNING _TYPE
			(BD/数据)		
-	PD Word0 的SourcePort[6:0]	判断为空包，丢弃，SN号不增加；	PD解析	有，连续失败次数达到上限则上报vf_get_poe_fail_rint（VF页签ACC_VF_ABNORMAL_INTERRUPT_SOURCE，偏移地址0x0208）	不涉及
-	PD Word0 的T_Len、T_TagValue、Msg_Len的TLV值错误	识别成错误PD，上报错误中断，丢弃任务，但不释放buf	PD解析	有，vf_get_poe_no_sec_task_rint中断（VF页签ACC_VF_ABNORMAL_INTERRUPT_SOURCE，偏移地址0x0208）	不涉及



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
-	PD Word4的T_Len、T_TagValue的TLV值错误	忽略Hac_Occ_SC配置，当做Hac_Occ_SC==0处理	PD解析	无	不涉及
-	PD的SN号错误	挂死，上报vf_get_poe_sn_err_rint	PD解析	有，vf_get_poe_sn_err_rint（ PF页签ACC_PF_ABNORMAL_INTERRUPT_SOURCE，偏移地址0x0208 ）	不涉及
-	reorder Buffer内部队头的SN号等待超时	挂死，上报vf_get_poe_sn_err_rint	PD解析	有，vf_get_poe_sn_err_rint（ PF页签ACC_PF_ABNORMAL_INTERRUPT_SOURCE，偏移地址0x0208 ）	不涉及
-	读BD时发生了rresperr	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	读BD	对应有1.（ PF 页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（ VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x1



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
-	选择了非法，或不支持的算法（ TDES/MD5/SHA1 ）， 或BD选择了没有权限的KEY。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1.（ PF 页签 ） SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint域段。2.（ VF页签 ） ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x2
0	BD_Type 不等于1或2或3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1.（ PF 页签 ） SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint域段。2.（ VF页签 ） ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x3



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
1	BD_Type 等于1但scene!=5	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x3
2	BD_Type 等于1但scene!=有效场景	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x3



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
3	BD1 AI_GEN>0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3
4	BD1，AUTH=2	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
5	BD1, CI_GEN=1/2	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3
6	BD1, CI_GEN>3	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
7	BD1 GMAC	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3
8	PBKDF2来自业务类型0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x3



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
9	Auth==3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
10	DE>1	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
11	Src_Addr_Type >=3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
12	Dst_Addr_Type >=3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
13	MAC_Addr_Type >=3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
14	Cipher_Key_Type!=0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
15	Auth_Key_Type!=0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
16	Gran_num == 0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
17	C_S==3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
18	Deal_ESP_AH != 0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
19	AUTH_PAD >1	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
20	Cipher_Pad_Data_Type >3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
21	Cipher_Pad_Len_Field >2	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
22	Write_Frame_Len>=3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
23	Cipher_Pad_Type>3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（ VF页签）ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x4
24	Cipher==3，DE==0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（ VF页签）ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
25	^Cipher==1 , CCM/GCM/ GMAC算法 时, ^Auth==1	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处 理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x5
26	^Cipher==1 , CCM/GCM 算法时, Auth_Sourc e_Offset+ Auth_Len> Cipher_Sou rce_Offset	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处 理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
27	^Cipher==1 , CCM/GCM 算法时, Auth_Source_Offset+Auth_Len+32<Cipher_Source_Offset	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x5
28	^Cipher==1 时, C_Alg, C_Mode, Ckey_Length, C_Width, C_ICV_LEN 域值组合不满足表1-22 描述	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
29	BD2, Cipher! =0 时, 非 gcm, Cipher_Length (加工后的) ==0 (gcm算法除外)	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x5
30	^Cipher==1 时, 非TLS/DTLS, 选择AES-CBC AES-ECB算法时 Cipher_Length非 16Byte对齐; (TLS 场景用补齐后的长度)	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
31	^Cipher==1时，非TLS/DTLS，选择DES/3DES算法时 Cipher_Length非8Byte对齐（TLS场景用补齐后的长度）	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1.（PF页签） SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（VF页签） ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
32	^Auth==1时，A_Alg，AKey_Length，Mac_Length域值组合不符合表1-21描述	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1.（PF页签） SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（VF页签） ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
33	BD2, 非gcm, ^Cipher=1, cipher_len (加工后)=0;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
34	Auth==2,G MAC算法时, AI_GEN==2	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
36	^Auth==1, AUTH_PAD=1 时,AI_GEN>1	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
37	^Cipher==1, GCM/CCM 算法时, Auth_Len=0, 且 Cipher_Len (加工后) ==0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
38	^Cipher==1，XTS模式 HUK=1	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
39	BD2，^Cipher==1，XTS模式，cipher_len (加工后) <16	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
40	BD2, ^Cipher==1, CTS模式, cipher_len (加工后) <17	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
41	^Cipher==1, CTS模式, bd_c_width == 3b000 或 >3b11	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
42	^Cipher==1，非CTS模式，bd_c_width != 3b000	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5
43	非存储上下盘场景 (BD1)，进算法核的数据大于24 ‘hff_ffff’ auth!=0, auth_ofst+auth_len+mac_len>32hff_ffff或 cipher!=0, cipher_ofst+cipher_len+mac_len>32hff_ffff (DTLS/TLS使用Padding需要再加上padding的长度)。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x5



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
44	非TLS/DTLS/存储上下盘， ^Cipher==1， ^Auth==1时,Cipher_Source_Offset<Auth_Source_Offset	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x6
45	非TLS/DTLS/存储上下盘， ^Cipher==1， ^Auth==1时,Cipher_Source_Offset+Cipher_Len>Auth_Source_Offset+Auth_Len	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x6



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
46	IPSEC, 非 ECB, ^Cipher==1, CI_GEN==1 时, IV_Offset+IV_len> Cipher_Source_Offset	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
47	IPSEC, ^Auth==1, AI_GEN==2/3, GMAC 算法时, IV_Offset+IV_len> Auth_Source_Offset+Auth_Len	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
48	IPSEC, ^Auth==1, AI_GEN==2/3, GMAC 算法时, IV_Offset<Auth_Source_Offset	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
49	C_S==1, Cipher==2	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
50	C_S==1, Auth==2	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x7
51	C_S==1, Cipher==0/3, Auth==0 时, udp_offset+8 > Cipher_Source_Offset+Cipher_Length	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
52	C_S==1, CIPHER==1 时, udp_offset +8 > CIPHER_SOURCE_Offset +CIPHER_Len	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
53	C_S==1, ^Auth==1 时, udp_offset +8 > Auth_Source_Offset +Auth_Len	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
54	C_S==1, CIPHER==1 时, CIPHER_LEN < 8 (UDP 头=8)	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
55	C_S==1, ^Auth==1 时, Auth_LEN< 8	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
56	C_S==2, Cipher==1 ,	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x7
57	C_S==2, Cipher==0/3, Auth==0 时, udp_offset +8 > Cipher_Sou rce_Offset +Cipher_Le n	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
58	C_S==2, CIPHER==2 时, CIPHER_LEN < 8	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
59	C_S==2, CIPHER==2 时, udp_offset +8 > CIPHER_SOURCE_Offset +CIPHER_LENGTH	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
60	C_S==2, ^Auth==1 时, udp_offset+8 > Auth_Source_Offset+Auth_Len	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7
61	C_S==2, ^Auth==1 时, Auth_Len<8	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0x7



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
62	^C_S==1 时，CS_UDP_Header_Offset - CS_IP_Header_Offset <40	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x7
63	存储上下盘，^Cipher==1，XTS模式，ci_gen!=3或非XTS模式， ci_gen==3	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
64	存储上下盘，auth==2	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x8
65	存储上下盘，配置与Block_Size不是九种支持的格式	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。3. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
66~73	存储上下盘， bd_block_type gen_page_pad_ctrl gen_grd_ctrl gen_ver_ctrl gen_app_ctrl gen_ref_ctrl page_pad_type chk_grd_ctrl chk_ref_ctrl 配置了缺省值	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。4. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8
74	存储上下盘， bd_gran_num*(bd_gran_size)>26h1ff_ffff	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。5. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
75	存储上下盘，选择了CCM GCM算法	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。6. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8
76	存储上下盘，选择了GMAC算法	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。7. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
77	存储上下盘，cipher_src_offset不为0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。8. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8
78	存储上下盘，auth_src_offset不为0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。9. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
79	PBKDF2场景，PBKDF2_Salt_Len<8;	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。10. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9
97	PBKDF2场景，DK_Len>512	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。11. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
80	PBKDF2场景， C_Num=0	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。12. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9
81	PBKDF2场景， pass_word_len < 1或 pass_word_len > 16h80	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。13. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
82	PBKDF2场景，auth = 2	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。14. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9
100	PBKDF2场景，Cipher!=0;	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。15. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
101	PBKDF2场景，选择了非HMAC类算法	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。16.（ VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9
110	PBKDF2，pass_word_len<1或非sha512/sha512_224/sha512_256/sha384 pass_word >64（该条检测关闭）	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。17.（ VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x9



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
83	非流模式非 gmac, Auth==2 时, ai_gen=1,AUTH_PAD=1	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。18. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0xA
84	^Cipher=1, ^Auth=1, AES_XTS模式, AI_GEN=1	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint 域段。19. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0xA



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
85	非流模式非gmac, sha160/ sha224/ sha256/ md5, Auth=1 时, AUTH_PAD=1, AI_GEN=1, (长数据HASH) Auth_Len[5:0]!=6h0;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。20. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xA
98	非流模式, 选中gmac算法时, ai_gen=0;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xA
	非流模式, 非gmac算法时, ai_gen>1;				
	非流模式, ^cipher=1, ai_gen>0;				



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
86	长数据HASH, Auth==1, AUTH_PAD==1时, 或 sha384、sha512、sha512/256、sha512/224类的算法 Auth_Length低7bit不为0;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0xA
99	auth=1, ai_gen=1,auth_pad=1时, 选中gmac或AES_XCBC或AES_CMAC算法	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 bd_err_rint域段。3. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的 vf_bd_err_rint域段	0xA



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
87	TLS auth_len != 5	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。4. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
88	DTLS auth_len != 13	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。5. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
102	DTLS/TLS 场景， ^cipher=1，非ccm非gcm非CBC	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（PF页签）SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。6.（VF页签）ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
103	DTLS/TLS 场景， cipher=1，cipher_pad_data_type=0,AES/SM4 -CBC算法，cipher_len非16B对齐；	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（PF页签）SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（VF页签）ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
	DTLS/TLS 场景， cipher=1，cipher_pad_data_type=3,AES/SM4 -CBC算法，cipher_len非16B对齐；				



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
104	DTLS/TLS 场景, cipher=1, cipher_pad_data_type=0,TDES/DES算法, cipher_len非8B对齐;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
	DTLS/TLS 场景, cipher=1, cipher_pad_data_type=3,TDES/DES算法, cipher_len非8B对齐;				
105	DTLS, cipher=1, cipher_offset< 认证头长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
106	DTLS, cipher=1, auth=1, cipher_offset< auth_offset + 认证头长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。3. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
107	DTLS, cipher=2, DES/TDES, cipher_offset< 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。4. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
108	DTLS, cipher=2, AES, cipher_offset< 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。5. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
109	DTLS, cipher=2, auth=2, DES/TDES, cipher_offset< auth_offset + 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。6. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
110	DTLS, cipher=2, auth=2, AES, cipher_offs et< auth_offset + 认证头长度 + IV长度;	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。7. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xB
111	TLS, cipher=1, cipher_offs et< 认证头长度;	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。8. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
112	TLS, cipher=1, auth=1, cipher_offset< auth_offset + 认证头长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。9. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
113	TLS, cipher=2, DES/TDES, cipher_offset< 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。10. (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
114	TLS, cipher=2, AES, cipher_offset< 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。11. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB
115	TLS, cipher=2, auth=2, DES/TDES, cipher_offset< auth_offset + 认证头长度 + IV长度;	上报中断, 回写BD ICV域置3, ERR_TYPE域段编码, 对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。12. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xB



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
116	TLS, cipher=2, auth=2, AES, cipher_offs et< auth_offset + 认证头长度 + IV长度;	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。13. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xB
89	流模式, gmac, auth_pad= 0;	上报中断, 回写BD ICV 域置3, ERR_TYPE 域段编码, 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xC
	流模式下, 选中gmac 算法时, ai_gen! =0;				
	流模式下, 非gmac算 法时, auth! =0b0;				



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
90	流模式， gmac， auth_len=0	上报中断， 回写BD ICV 域置3， ERR_TYPE 域段编码， 对报文不处理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。2. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xC
91	流模式， gmac， cipher_len (加工后) >0	上报中断， 回写BD ICV 域置3， ERR_TYPE 域段编码， 对报文不处理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。3. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xC



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
92	流模式，gcm算法，auth_pad=1,cipher_len非16B对齐。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。4.（ VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xC
93	流模式，gcm算法，auth_len!=0;	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（ PF 页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。5.（ VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xC



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
94	流模式， gcm算法， auth_offset != cipher_offset	上报中断， 回写BD ICV域置3， ERR_TYPE域段编码， 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。6. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xC
95	流模式， gcm算法， cipher_len=0	上报中断， 回写BD ICV域置3， ERR_TYPE域段编码， 对报文不处理。	BD解析	对应有1. (PF 页签) SEC_PF_AB_NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。7. (VF页签) ACC_VF_AB_NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xC



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
96	流模式， gcm算法， ai_gen=0， auth_pad= =0	上报中断， 回写BD ICV 域置3， ERR_TYPE 域段编码， 对报文不处 理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。8. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xC
118	流模式非 gcm非 gmac， auth_pad= 1；	上报中断， 回写BD ICV 域置3， ERR_TYPE 域段编码， 对报文不处 理。	BD解析	对应有1. (PF 页 签) SEC_PF_AB NORMAL_I NT_SOURC E寄存器的 bd_err_rint 域段。9. (VF页签) ACC_VF_AB NORMAL_I NT_SOURC E寄存器的 vf_bd_err_r int域段	0xC



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
112	读BD时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。10. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xc
128	读KEY时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应1. (PF 页签) SEC_PF_AB NORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。11. (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0xF



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
129	读IVIN时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	BD解析	对应有1.（PF页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。12.（VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段	0x10
130	读数据方向读链表时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	总线异常	对应SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的axi_rresp_err_rint/chain_buff_err_rint域段。	0x11
131	读数据方向读链表时，发生解析异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	链表异常	对应SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的chain_buff_err_rint域段。	0x12
132	读报文时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	总线异常	对应SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的axi_rresp_err_rint域段。	0x13



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
133	写数据方向读链表时，发生总线异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	总线异常	对应SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的axi_rresp_err_rint域段。	0x14
134	写数据方向读链表时，发生解析异常。	上报中断，回写BD ICV域置3，ERR_TYPE域段编码，对报文不处理。	链表异常	对应SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的axi_rresp_err_rint/chain_buff_err_rint域段。	0x15
135	BD3，Src_Addr_Type=3（即单节点PPE链），NextAddr=0时，data_len为0或者RefItselfAddr为0	识别成错误BD，上报BD错误中断，回写BD时ICV域置3，对报文不进行处理	BD解析	对应有1.（PF页签）SEC_PF_ABNORMAL_INTERRUPT_SOURCE寄存器的bd_err_rint域段。2.（VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_bd_err_rint域段或3.（VF页签）ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_chain_buff_err_rint域段	0x8



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
136	BD3, Src_Addr_Type=3 (即 PPE链) , data_len_all<SEC计算所需要的报文长度	读方向获取链表的第一个非0长度节点地址读取剩下所需长度报文, 写方向不够部分报文发出空写操作。	链的解析	对应有VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_chain_buff_err_rint 域段1	0x15
137	BD3, Src_Addr_Type=3 (即 PPE链) , Buffer链中所有的 Data_Length长度都是0	可能挂死, 如果不挂死则BD回写时会识别成错误BD, 上报链表节点全0中断和BD错误中断, ICV域置3	链的解析	对应有 (VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的会上报 vf_chain_buff_len_all_0_rint; 如果不挂死, 则有 vf_chain_buff_err_rint 。	0x15
138	BD3, Src_Addr_Type=3 (即 PPE链) , 首节点链表的 NextDataAddr域低6bit不为0	识别成错误BD, 上报BD错误中断, 回写BD时ICV域置3, 对报文不进行处理	BD解析	对应有VF页签) ACC_VF_AB NORMAL_INTERRUPT_SOURCE寄存器的 vf_chain_buff_err_rint 域段1	0x15



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
139	BD3, Src_Addr_Type=3 (即PPE链), 链表的NextDataAddr域低6bit不为0	出错节点当最后一个节点, 然后和正常链表处理一样, 如果此时链表长度不够, 读方向获取链表的第一个非0长度节点地址读取剩下所需长度报文, 写方向不够部分报文发出空写操作。	链的解析	对应有 (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_chain_buff_err_rint域段	0x15
1140	BD3, Src_Addr_Type=3 (即PPE链), PPE链个数超过寄存器配置的PPE链阈值	可能挂死, 如果不挂死则上报链表异常中断	链的解析	对应有 (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_chain_buff_err_rint域段	0x15
141	链表非首节点, RefltselAddr==0	出错节点的报文长度当0处理, 然后和正常链表处理一样, 如果此时链表长度不够, 读方向获取链表的第一个非0长度节点地址读取剩下所需长度报文, 写方向不够部分报文发出空写操作。	链的解析	对应有 (VF页签) ACC_VF_ABNORMAL_INTERRUPT_SOURCE寄存器的vf_chain_buff_err_rint域段	0x15



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
142	C_S==1 , udp_offset +udp_len>c hain_len	回写BD ICV 域置3, ERR_TYPE 域段编码。	checksum 数据异常	无中断	0x16
	注: chain_len 为 auth_offset +auth_leng th +mac_lengt h 或				
	cipher_offs et +cipher_len gth				
	(如果同时 有两种运 算, 选取较 大的那个 值)				
1143	C_S==2, Cipher==0/ 3, Auth==0 时, udp_offset +udp_len > Cipher_Sou rce_Offset +Cipher_Le n	继续处理, 仅CSC可能 为2	BD解析	无中断	0x16
144	CHKSUM报 文错误	回写BD ICV 域置3, ERR_TYPE 域段编码。	BD解析	无中断	0x16



bd_err_chk_en对应bit	出错信息描述	处理策略	出错位置	是否有单独原始中断对应	ERROR_TYPE/ WARNING_TYPE
			(BD/数据)		
145	C_S==2, Cipher==2 时, udp_offset + udp_len > Cipher_Source_Offset + Cipher_Len	回写BD ICV 域置3, ERR_TYPE 域段编码。	BD解析	无中断	0x16
146	Unwrap key 解密出错	回写BD ICV 域置3, ERR_TYPE 域段编码。	BD解析	无中断	0x17
-	C_S==2, ^ Auth==1 时, udp_offset +udp_len > Auth_Source_Offset +Auth_Len ; C_S==2, ^ Auth==1 时, checksum校验失败。	继续处理, 仅CSC可能为2	BD解析	无中断	Waening

表 8-17 未检测异常 BD 处理

未检测异常BD					
1	C_S==1时, udp_len<8	软件规避	checksum	无	0x18
2	C_S==2时, udp_len<8	继续处理, 仅CSC可能为2	checksum	无	报warning为0x1
3	DTLS场景, cipher=1,auth=2	软件规避	-	-	驱动禁止这种配置



未检测异常BD					
4	SSL/TLS场景， cipher=1,auth=2	软件规避	-	-	驱动禁止这种配置

8.8.2 PD MEM 和 BD MEM 可视化

软件可以通过寄存器接口，查询SEC内部缓存PD的SRAM和缓存BD的SRAM中的内容；该DFX功能是为了方便原型验证、样片测试等非EDA场景的调试（商用时禁止使用该接口，以免引起不必要的错误中断）。

受限与SRAM的特性，需要在SEC处于空闲阶段再使用SRAM可视化，否则可能导致查询出非预期结果（甚至可能伴随错误中断上报）。

PD SRAM可视化具体使用方式：

- 步骤1 配置SEC_POE_REG页签SEC_POE_MEM_RD_CFG（偏移地址为0xCCC）。
- 步骤2 通过读SEC_POE_REG页签SEC_POE_MEM_RDATA（偏移地址为0XCD0~0xCFC）获得步骤1查询的BD内容。这8个寄存器会根据步骤1的配置而更新。

须知

只有查询和配置操作相隔至少20拍AXI Slave时钟，才能查询出正确结果。

----结束

BD SRAM可视化具体使用方式：

- 步骤1 配置PF SEC_BDMEM_RD_CFG（偏移地址为0xCCC）。
- 步骤2 通过读PF SEC_BDMEM_RDATA（偏移地址为0XCD0~0xCFC）获得步骤1查询的BD内容。这8个寄存器会根据步骤1的配置而更新。

须知

只有查询和配置操作相隔至少20拍AXI Slave时钟，才能查询出正确结果。

----结束

非预期结果排除：

- 受限与SRAM处理机制，在SEC正常工作过程中，配置寄存器可视化操作可能会和SEC正常功能逻辑有冲突，此时查询结果有效状态位会置0。
- 在SEC工作期间进行查询操作，因为SEC在内部不断地存取PD、BD，所以即便是查询8个连续的地址，也可能查出的是不属于同一个PD、BD的内容。

8.8.3 典型定位案例

NA



8.9 注意事项

无



9 TRNG

9.1 概述

True Random Number Generator (TRNG) 是用来产生真随机数的硬件单元，通过硬件通道提供随机数给各个加速器，CPU通过总线接口获取随机数。

图 9-1 TRNG 应用框图

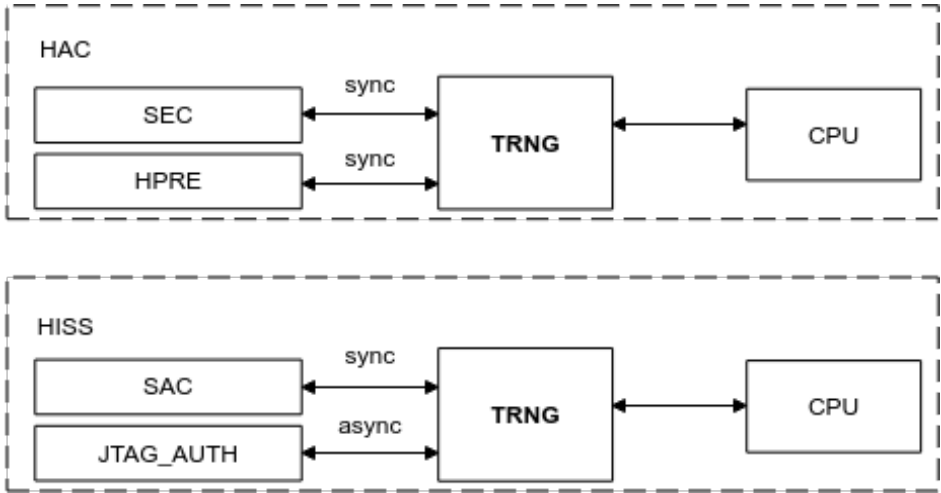
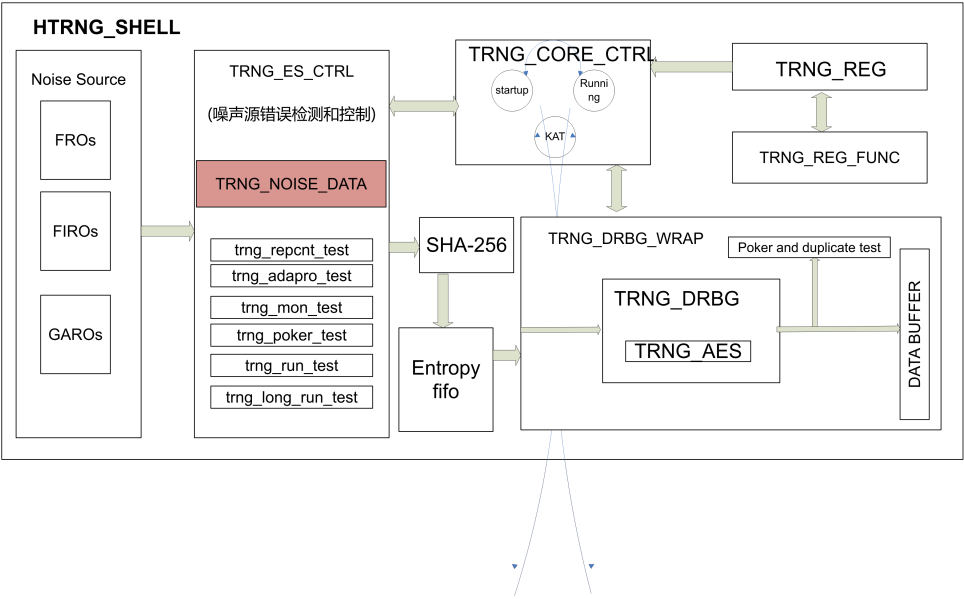




图 9-2 HTRNG IP 的内部逻辑框图



9.2 关键术语

表9-1为与TRNG相关的关键术语表。

表 9-1 TRNG 关键术语描述

术语	描述
TRNG	True Random Number Generator，真随机数产生器。
NRBG	Non-deterministic Random Bit Generator，不确定性随机bit产生器。
DRBG	Deterministic Random Bit Generator，确定性随机bit产生器。
NIST	National Institute of Standards and Technology，国际标准和 技术协会。
FIPS	Federal Information Processing Standard。
FRO	Free Running Oscillator。
FIRO	Fibonacci Ring Oscillator，一种类似于Fibonacci型非线性反馈 移位寄存器的环形振荡器。
GARO	Galois Ring Oscillator，一种类似于Galois型非线性反馈移位寄 寄存器的环形振荡器。
Repetition Count	重复性计数。
Adaptive Propotion	自适应均衡。



术语	描述
Monobit test	测试1和0的数目是否相等。
Poker test	测试不交迭4bit block的出现。
Runs test	测试0或者1的数目，长度小于6或者大于等于6。
Long runs test	搜索长度大约等于34。
Noise source failure	噪声源失败测试。
AIS	Application notes and Interpretation of the Scheme (German IT standard series)。
AES	Advanced Encryption Standard。
ANSI	American National Standards Institute。
CTR_DRBG	Counter-Deterministic Random Bit Generator (SP 800-90A classification for EIP-76d DRBG)。
IV	Initialization Vector。
SHA	Secure Hash Algorithm。
KAT	Known Answer Testing, 结果已知的测试，即已知正确的输出结果，检查经过计算的输出结果是否和已知正确的输出结果相等，相等即测试通过，否则测试失败。
Reseed	给DRBG的输入提供一些随机值，DRBG根据输入的随机种子进行随机数的产生。

9.3 特性

TRNG具有以下特性：

- 支持通过硬件通道提供随机数给硬件加速器。
 - 通过同步硬件接口通道提供随机数给SAC、HPRE、SEC密码算法核加速器。
 - 通过异步硬件接口通道提供随机数给JTAG_AUTH。
- TRNG硬件实现包括不确定性随机bit生成器（NRBG）与确定性随机bit生成器（DRBG）。
- NRBG中包含硬件加工功能：
 - 加工函数可配置选择采用SHA-256算法（[SP 800-90B]（和[FIPS140-2]/[FIPS140-3 draft]））或者SM3算法（[GM/T 004-2012 SM3 密码杂凑算法]）。
 - 加工功能把前面的输出追加到噪声源输入，平滑噪声源行为的变化。
- 支持确定性随机bit生成器（DRBG）：
 - 与AES-256的CTR_DRBG兼容（[SP 800-90A]（[FIPS 140-2]/[FIPS 140-3]）（[GM/T 002-2012 SM4 分组密码算法]）），不带DC_DF功能。



- DRBG基于一次种子产生的随机数个数可配置（1~4096）。
- DRBG功能可以被关闭，DRBG功能被关闭后，最终输出的随机数为加工函数产生的随机数。
- DRBG随机数数据缓存Buffer，最大16Kbits深度（采用单口SRAM，最大为512X（32+crc4）bits）。
- 支持物理噪声源。
 - 支持4条长度不同的FRO环形振荡器，可单独配置使能。
 - 支持4条长度不同的FIRO环形振荡器，可单独配置使能。
 - 支持4条长度不同的GARO环形振荡器，可单独配置使能。
 - 支持4条长度不同的FRO环形振荡器，作为TRNG的内置时钟。
 - 物理噪声源最终异或输出为一路。
 - 支持对采样的噪声源数据做异或压缩，异或压缩的次数可配置为1~65535，默认不压缩。
- 支持随机数在线健康检查测试：
 - 在NRBG和DRBG上的重复输出数据的检测（[FIPS 140-2][FIPS 140-3 draft]）。
 - 硬件实现在噪声源的“重复计数（Repetition Count）”和“自适应均衡（Adaptive Propotion）”测试，测试窗口为1024大小（NIST SP800-90B）。
 - 支持对噪声源的连续测试（[AIS-31]）：
 - Monobit test。
 - Poker test。
 - Runs test。
 - Long runs test。
 - 支持在加工功能中DRBG和自测试电路上各种在线和离线的完整测试，以及KAT（known-answer tests）。
 - 支持startup测试，测试通过之后才能产生随机数。
- 熵采样周期可以配置。
- 支持硬件自动re-seed功能，不需要软件协助做re-seed，re-seed的数据产生间隔软件可配置。
- 支持硬件自动初始化，不需要软件初始化TRNG。
- 随机数产生性能：
 - TRNG DRBG随机数产生8bits/clock。
- TRNG系统安全特性。
 - TRNG内部寄存器只能安全可配置。
 - TRNG中断为安全中断。
- TRNG不能作为MASTER。
- TRNG作为Slave。
- 支持故障检测上报
 - TRNG健康检测，支持中断上报。



- 支持故障检测统计。
- 支持中断屏蔽。
- 支持APB接口，用于软件配置和随机数获取。
- 支持DFT测试振荡环。

9.4 遵循协议

NRBG：与SHA-256兼容（[SP 800 -90B]（和[FIPS140-2]/[FIPS140-3 draft]））；

DRBG：与AES-256的CTR_DRBG兼容（[SP 800-90A]（[FIPS 140-2]/[FIPS 140-3] draft））；

在NRBG和DRBG上的重复输出数据的检测（[FIPS 140-2][FIPS 140-3 draft]）；

噪声源的连续测试（[AIS-31]）。

9.5 功能描述

9.5.1 随机数生成

TRNG按照SP800-90A/90B标准进行设计，随机数的生成包括NRBG和DRBG两部分。真随机数的随机性，依赖于物理噪声源，通过加工函数产生全熵输出。DRBG利用NRBG提供的全熵数据作为种子，产生随机数数据。

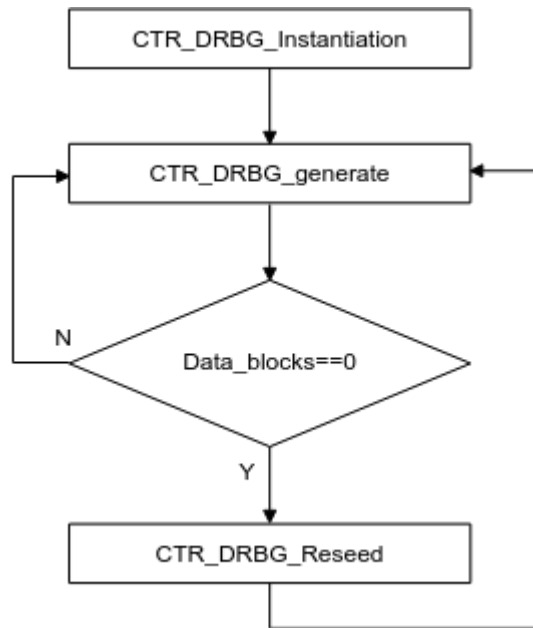
9.5.1.1 NRBG

NRBG即Non-deterministic Random Bit Generator，是通俗理解的基于物理随机现象产生的随机数。在本设计通过采样环形振荡器的随机抖动作为熵源，然后通过SHA256/SM3算法对噪声源数据进行加工产生全熵输出，提供DRBG所需的种子。

9.5.1.2 DRBG

DRBG即Deterministic Random Bit Generator，确定性随机数生成器。DRBG基于特定的密码算法，通过提供的种子和初始状态来产生随机输出。在NIST SP800-90A标准中，基于Hash函数，DRBG有Hash_DRBG和HMAC_DRBG这两种生成方式。基于分组密码算法，有CTR_DRBG的随机数生成方式。在本设计采用基于AES-256的CTR_DRBG生成随机数。

图 9-3 CTR_DRBG 算法流程



9.5.2 随机数测试

按照NIST SP800-90B、AIS-31标准，TRNG内部硬件电路会随机数进行StartUp测试和连续性在线测试。包括NIST SP800-90A规定的重复性计数测试（Repetition Count Test）和自适应比例测试（Adaptive Proportion Test）。AIS-31规定的单位元检测（The monobit test）、扑克检测（The Poker test）、游程测试（The runs test）、长游程测试（The long runs test）以及噪声源失效测试（Noise Source failure test）。

在测试模式下，可以通过软件配置完成对各功能电路（如，SHA256，AES，CTR_DRBG）的KAT（Known Answer Test）测试。同样在测试模式下，通过配置软件可以直接通过读取寄存器TRNG_NOISE_L/H来获取噪声源数据，从而对TRNG噪声源的熵值进行评估。

9.5.3 随机数性能

TRNG中DRBG随机数产生性能为8bits/clock。在800M工作时钟下面随机数产生速率为800MBps。

9.6 软件指南

9.6.1 复位和初始化

9.6.1.1 复位

TRNG上电时默认处于解复位状态，上电时钟默认使能。支持软件复位。



9.6.1.2 硬件初始化

TRNG支持解复位之后，自动进行随机数的产生。硬件初始化的参数是固定的，不需要软件配置，即可产生随机数。

9.6.1.3 软件初始化

- 步骤1** 配置寄存器TRNG_CONTROL，写全0，确保HTRNG处在一个确定状态。
- 步骤2** 配置HTRNG_INT_SOURCE为0xFFFF，清除起始的无效中断。
- 步骤3** 配置寄存器HTRNG_SAMPLE_CFG为0x00000003，设置噪声源采样间隔周期。HTRNG_SAMPLE_CFG.sample_div为FROs采样的时钟间隔，0表示每个时钟都从FROs采样，15表示每16个时钟周期采样一次。设置HTRNG_SAMPLE_CFG.sample_cycles，设置异或到一起生成1bit的FRO采样bit次数，异或结果再移位到主移位寄存器中。如果测试到的噪声源数据，随机效果不理想，则增大sample_div和sample_cycles的值，有益于**提高熵值**。配置寄存器HTRNG_CONFIG为0x000FFF08。这个设置表示每次需要8个512bit的噪声源数据blocks用于SHA256加工函数输出256bit输出，每一轮DRBG产生4096个block（128bit）随机数。NIST SP800-90B要求经过加工函数产生全熵输出。对于256bit的全熵输出，要求输入给加工函数的熵值为512bit（2倍输入熵），假设每8bits采样含有1bit熵，则需要8个512bit噪声源数据块。
- 步骤4** 配置寄存器HTRNG_INT_ENABLE，打开相应的中断。
- 步骤5** 配置寄存器HTRNG_NOISE_ENABLE为0x00000FFF，打开4个FRO环，4个Fibonacci环，4个Galois环。
- 步骤6** 配置寄存器HTRNG_CONTROL为0x00000E3，使能AIS31，SP800的online测试，使能DRBG和trng_enable，开始随机数的产生。

----结束

9.6.2 应用编程接口

9.6.2.1 测试 AIS-31 Monobit,run,poker 测试电路操作流程

- 步骤1** 设置寄存器HTRNG_CONTROL，写全0，确保TRNG处在一个确定状态。
- 步骤2** 设置HTRNG_TEST.test_mode=1，HTRNG进入测试模式。设置HTRNG_CFG_START_UP.start_up_bypass=1。HTRNG_CONTROL.ais31_tst_en=1，设置HTRNG_TEST.test_noise=1。开始进行AIS-31的KAT测试。
- 步骤3** 查询HTRNG_STATUS.test_noise_ready，为1表示可以写入一个64bits测试数据。写入测试数据到HTEST_NOISE_L/H，然后置位HTRNG_TEST.test_noise_vld=1，表示64bits噪声源测算数据写入完成。
- 步骤4** 重复**步骤3**的数据输入。根据输入的测试数据，软件查询HTRNG_INT_STATUS是否有相应的异常产生（需要打开中断开关）。
- 步骤5** 清除HTRNG_TEST.test_mode，HTRNG_TEST.test_noise，HTRNG_CFG_START_UP.start_up_bypass退出测试模式。

----结束



须知

检测时是以128bits数据为单位进行Monobit, run和poker检测。例如, 将HTRNG_ALARM_THRESHOLD寄存器中的域值设置成1, 那么至少要连续输入两个异常128bits数据块才能产生异常, 也就是**步骤3**至少要重复进行4次。

9.6.2.2 测试 repetition count, Adaptive Propotion 测试电路操作流程

- 步骤1** 设置寄存器HTRNG_CONTROL, 写全0, 确保TRNG处在一个确定状态。
- 步骤2** 设置HTRNG_TEST.test_mode=1, HTRNG进入测试模式。设置HTRNG_CFG_START_UP.start_up_bypass=1。设置噪声源检查报警阈值寄存器HTRNG_ALARM_THRESHOLD。HTRNG_CONTROL.sp800_tst_en=1, HTRNG_TEST.test_spb=1。开始进行repetition count, Adaptive Propotion的KAT测试。
- 步骤3** 查询HTRNG_STATUS.test_noise_ready, 为1表示可以写入一个64bits测试数据。写入测试数据到HTEST_NOISE_L/H, 然后置位HTRNG_TEST.test_noise_vld=1, 表示64bits噪声源测算数据写入完成。
- 步骤4** 重复**步骤3**的数据输入。根据输入的测试数据, 软件查询HTRNG_INT_STATUS是否有相应的异常产生(需要打开中断开关)。
- 步骤5** 清除HTRNG_TEST.test_mode, HTRNG_TEST.test_spb, HTRNG_CFG_START_UP.start_up_bypass, 退出测试模式。

----结束

须知

检测时repetition count测试是检测连续重复数据bit的个数, 如果连续重复数据bit个数超过HTEST_SPB_CTRL.repcnt_cutoff的值, 则会产生一次失败, 产生失败的次数如果超过了HTRNG_ALARM_THRESHOLD寄存器中设置的次数, 则会上报告警。Adaptive Propotion测试电路统计1024bits数据窗口里面的数据, 统计1024bits数据窗口里面和首次采样到的bit的相同bit的个数, 超过阈值则产生一次失败, 产生失败的个数超过HTRNG_ALARM_THRESHOLD里面设置的个数, 则上报告警。

9.6.2.3 测试 SHA256/SM3 加工功能操作流程

- 步骤1** 等待HTRNG_FIFO_STATUS.entropy_fifo_full=1和HTRNG_FSM_ST.fsm_state=7。设置HTRNG_CONTROL为全0, 关闭sp800, AIS31噪声源在线测试, 关闭DRBG使能HTRNG_CONTROL.drbg_enable=0, 关闭HTRNG使能。设置HTRNG_NOISE_ENABLE为全0, 关闭噪声源。配置HTRNG_ALG_MODE.hash_md选择测试SHA256算法或者SM3算法。
- 步骤2** 设置HTRNG_CONFIG.noise_blocks为将要测试输入的噪声源数据块数量, 每个噪声源数据块为512bits。HTRNG的加工函数输入还包括另外256bits的optional additional input, 按照SP800-90B, 正常模式下optional additional input是上一次加工函数的输出。依次写入32bits测试数据到HTRNG_DRBG_PS_AI_0~7, 作为optional additional input, 即最后一次Hash运算的输入。



- 步骤3** 设置HTRNG_TEST.test_mode=1，HTRNG进入测试模式。然后设置HTRNG_TEST.test_cond_func=1，HTRNG进入SHA256加工功能的KAT测试。
- 步骤4** 查询HTRNG_STATUS.test_ready，为1表示可以进行噪声源数据输入，依次写入32bits测试数据到HTEST_NOISE_H、HTEST_NOISE_L，然后置位HTRNG_TEST.test_noise_vld，表示已经成功写入64bits测试数据。连续进行8次写入64bits数据的操作，输入一次512bits的noise数据块。
- 步骤5** 根据设置的HTRNG_CONFIG.noise_blocks大小，重复**步骤4**的数据输入。
- 步骤6** 测试数据输入完毕之后，查询HTRNG_STATUS.test_ready，为1表示SHA256加工函数测试完毕。读取HTRNG_CON_DATA_0~7即最后输出的加工函数KAT测试结果。
- 步骤7** 清除HTRNG_TEST.test_cond_func，清除HTRNG_TEST.test_mode，退出测试模式。

----结束

下面是一个针对SHA-256的标准的测试向量。

图 9-4 测试向量

Length= 768

Message=

73747576727374756E6F70716D6E6F706C6D6E6F6B6C6D6E
6A6B6C6D696A6B6C68696A6B6768696A66676869656667686
465666763646566626364656162636400000000000000010000
00020000000300000004000000050000000600000007

图 9-5 写入加工函数的 optional additional input

TRNG_DRBG_PS_AI_0 = 0x00000000

⋮

TRNG_DRBG_PS_AI_7 = 0x00000007

图 9-6 写入噪声源测试数据

TRNG_DRBG_PS_AI_0 = 0x00000000

⋮

TRNG_DRBG_PS_AI_7 = 0x00000007



SHA256_MD= e98f5aee878c8e28af7b425795ed5a2c17872

53111fc44404381585d05551ff5

SM3_MD=C2F3EC43853BB12036FB87E2B6C37A880FEBEB4BEEAED2C1ACDBC0E40
D2B54C9

9.6.2.4 测试 SP 800-90A DRBG-AES/SM4_ECB_256 操作流程

- 步骤1** 设置HTRNG_CONTROL为全0，关闭sp800，AIS31噪声源在线测试，关闭DRBG使能，关闭HTRNG使能。设置HTRNG_NOISE_ENABLE为全0，关闭噪声源。设置HTRNG_ALG_MODE.drbg_alg_md选择AES或SM4算法测试。
- 步骤2** 配置HTRNG_INT_SOURCE为0xFFF，清除起始的无效中断（根据配置的DRBG数量，清除操作重复多次）。
- 步骤3** 设置HTRNG_TEST.test_mode=1，HTRNG_TEST.test_drbg=1。然后设置HTEST_DRBG_CTRL.test_aes_256=1，进入AES/SM4 ECB的测试模式。
- 步骤4** 写入测试key值到HTRNG_DRBG_PS_AI4~11，写入测试明文到HTRNG_DRBG_PS_AI0~3。
- 步骤5** 设置HTRNG_CONTROL.drbg_enable_0启动DRBG_0的测试。
- 步骤6** 查询HTRNG_STATUS.test_ready，为1表示测试完成。
- 步骤7** 读取HTRNG_DATA_0~3的数据，即AES_ECB_256的KAT测试结果。
- 步骤8** 清除HTRNG_CONTROL.drbg_enable。
- 步骤9** 如果需要测多次，则重复**步骤3~步骤7**。
- 步骤10** 清除HTRNG_TEST.test_mode，HTRNG_TEST.test_mode，HTEST_DRBG_CTRL.test_aes_256，退出测试模式。

----结束

下面是一个针对AES-ECB-256的标准的NIST测试向量。

图 9-7 NIST 测试向量

Plaintext= 7e10eadc228fdf487462a5848c1cf3a7

TRNG_DRBG_PS_AI_0 = 0x8c1cf3a7
TRNG_DRBG_PS_AI_1 = 0x7462a584
TRNG_DRBG_PS_AI_2 = 0x228fdf48
TRNG_DRBG_PS_AI_3 = 0x7e10eadc



图 9-8 写入测试 key 值到 HTRNG_DRBG_PS_AI4~11

Key =
48cc7c69dea37b4c25812ec701e0c5865adfc009835902a590ee46f6acf
c6035
TRNG_DRBG_PS_AI_4 = 0xacfc6035
TRNG_DRBG_PS_AI_5 = 0x90ee46f6
TRNG_DRBG_PS_AI_6 = 0x835902a5
TRNG_DRBG_PS_AI_7 = 0x5adfc009
TRNG_DRBG_PS_AI_8 = 0x01e0c586
TRNG_DRBG_PS_AI_9 = 0x25812ec7
TRNG_DRBG_PS_AI_10 = 0xde37b4c
TRNG_DRBG_PS_AI_11 = 0x48cc7c69

图 9-9 写入测试明文到 HTRNG_DRBG_PS_AI0~3

Ciphertext(AES)= d3ffe9c157e461fdc5729bbe113c387
TRNG_DATA_0 = 0xd113c387
TRNG_DATA_1 = 0xc5729bbe
TRNG_DATA_2 = 0x57e461fd
TRNG_DATA_3 = 0xd3ffe9c1

Ciphertext(SM4)= 44484cbf4e50d2a8ba7c33e325e65d10
TRNG_DATA_0 = 0x25e65d10
TRNG_DATA_1 = 0xba7c33e3
TRNG_DATA_2 = 0x4e50d2a8
TRNG_DATA_3 = 0x44484cbf

9.6.2.5 测试标准的 NIST SP 800-90A DRBG 操作流程

对于SP 800-90A DRBG的测试，包括Instantiaton、Generate、Reseed三个子流程的KAT测试。每个子流程都可以输出相应的测试结果。在FIPS提供的测试数据里面，对测试数据进行了区分，分为带reseed子过程和不带reseed子过程。TRNG PredictionResistance = True。



9.6.2.5.1 不带 reseed DRBG KAT 测试流程

- 步骤1** 设置HTRNG_CONTROL为全0，关闭sp800，AIS31噪声源在线测试，关闭DRBG使能，关闭TRNG使能。设置HTRNG_NOISE_ENABLE为全0，关闭噪声源。配置HTRNG_INT_SOURCE为0xffff，清除起始的无效中断（根据配置的DRBG数量，清除操作重复多次）。
- 步骤2** 设置HTRNG_TEST.test_mode=1，HTRNG_TEST.test_drbg=1。配置HTRNG_MODE为0x1，通过寄存器接口获取随机数模式。
- 步骤3** 设置HTRNG_CONFIG.data_blocks=4，写入384bit的向量（包含一个等于特征数据串异或熵输入的值）到寄存器HTRNG_DRBG_PS_AI_0~11。
- 步骤4** 设置HTEST_DRBG_CTRL.test_drbg_init=1，然后设置HTRNG_CONTROL.drbg_enable_0=1，开始进行DRBG的 Instantiation计算。
- 步骤5** 查询HTRNG_STATUS.test_ready，为1表示Instantiation子流程计算完成，读取HTRNG_DRBG_KEY_0~7，HTRNG_DRBG_V_0~3。
- 步骤6** 写入384bit的向量（第一次Generate的addition_input值），然后设置HTEST_DRBG_CTRL.test_drbg_gen=1，开始进行DRBG的第一次Generate子流程计算。
- 步骤7** 查询HTRNG_INT_STATUS.ready_int_status，为1表示Generate产生一个128bits的随机数。获取HTRNG_DATA_0~3的随机数据。设置HTRNG_INT_SOURCE.ready_int_source，清除随机数据ready中断状态位。
- 步骤8** 按照**步骤3**设置的data_blocks次数重复**步骤7**。标准的NIST SP 800-90A DRBG known-answer测试丢弃第一次Generate功能的结果。查询HTRNG_STATUS.test_ready，为1表示第一次Generate子流程计算完成。写入384bit的向量（第二次Generate的addition_input值）。开始进行DRBG的第二次Generate子流程计算。
- 步骤9** 查询HTRNG_INT_STATUS.ready_int_status，为1表示Generate产生一个128bits的随机数。设置HTRNG_INT_SOURCE.ready_int_source，清除随机数据ready中断状态位。
- 步骤10** 按照**步骤3**设置的data_blocks次数重复**步骤9**，获取HTRNG_DATA_0~3的数据之后和CAVP的测试向量进行对比。查询HTRNG_STATUS.test_ready，为1表示第二次Generate子流程计算完成。
- 步骤11** 清除HTRNG_CONTROL.drbg_enable0，然后清除HTRNG_TEST.test_mode，HTEST_DRBG_CTRL.test_drbg_init，HTEST_DRBG_CTRL.test_drbg_gen，HTRNG_MODE，退出测试模式。

----结束

下面是一个针对CTR_DRBG测试向量



图 9-10 CTR_DRBG 测试向量

```
EntropyInput =  
7bafbfa40862c36342312e2ba51525e517cf1edd8c2aeb62bd1109a2fbb526424305079f532834cadd894dc  
f7bb42e7e  
  
PersonalizationString =  
d91b3463219ccc204511ceee4861ffb1258b40d2cbf70977c295619e4a47c1e7bd377dc467d51dd437f24  
a2f944ec1f7  
  
TRNG_DRBG_PS_AI_0 = 0x7bb42e7e XOR 0x944ec1f7  
:  
:  
:  
TRNG_DRBG_PS_AI_11 = 0x7bafbfa4 XOR 0xd91b3463
```

第一次写入

图 9-11 第二次写入

```
EntropyInputPR =  
d7213f439cac915b49cbcb06971caa3d907aa77dad9ecc70dbf5be2140be5e8f822d4003d28e62978e112588f3a  
5416a  
  
TRNG_DRBG_PS_AI_0 = 0xf3a5416a  
:  
:  
:  
TRNG_DRBG_PS_AI_11 = 0xd7213f43
```

第二次写入

图 9-12 第三次写入

```
EntropyInputPR =  
eb6c35a3a229b33990a8d54f497178af181d645597cad32b87f86dfc17b5b5b9962ca256c31f665b3216b  
bf6e63af66d  
  
TRNG_DRBG_PS_AI_0 = 0xe63af66d  
:  
:  
:  
TRNG_DRBG_PS_AI_11 = 0xeb6c35a3
```

第三次写入

图 9-13 结果数据

```
ReturnedBits =  
36b7c45acd9db6d2beefd51ae4f2a06fefe9d26bbf9ce6351b7fa712538b28d366ccee428b012ebbd209fcb42a75  
979d6074301e5306ca20551fa8f025e98eb
```

生成的512bits DRBG数据



9.6.2.6 带 reseed DRBG KAT 测试流程

- 步骤1** 设置HTRNG_CONTROL为全0，关闭sp800，AIS31噪声源在线测试，关闭DRBG使能，关闭HTRNG使能。设置HTRNG_NOISE_ENABLE为全0，关闭噪声源。配置HTRNG_INT_SOURCE为0xFFF，清除起始的无效中断（根据配置的DRBG数量，清除操作重复多次）。
- 步骤2** 设置HTRNG_TEST.test_mode=1，HTRNG_TEST.test_drbg=1。配置HTRNG_MODE为0x1，通过寄存器接口获取随机数模式。
- 步骤3** 设置HTRNG_CONFIG.data_blocks=4，写入384bit的向量（包含一个等于特征数据串异或熵输入的值）到寄存器HTRNG_DRBG_PS_AI_0~11。
- 步骤4** 设置HTEST_DRBG_CTRL.test_drbg_init=1，然后设置HTRNG_CONTROL.drbg_enable_0=1，开始进行DRBG的 Instantiation计算。
- 步骤5** 查询HTRNG_STATUS.test_ready，为1表示Instantiation子流程计算完成，读取HTRNG_DRBG_KEY_0~7，HTRNG_DRBG_V_0~3。
- 步骤6** 写入384bit的向量（第一次Reseed的EntropyInput和AdditionInput的异或值），然后设置HTEST_DRBG_CTRL.test_drbg_reseed=1，开始进行DRBG的第一次reseed子流程计算。
- 步骤7** 查询HTRNG_STATUS.test_ready，为1表示第一次reseed子流程计算完成。设置HTEST_DRBG_CTRL.test_drbg_reseed=0，设置HTEST_DRBG_CTRL.test_drbg_gen=1。开始进行DRBG的第一次Generate子流程计算。
- 步骤8** 查询HTRNG_INT_STATUS.ready_int_status，为1表示128bits随机数据产生，设置HTRNG_INT_SOURCE.ready_int_source=1，清除随机数据ready中断状态位。
- 步骤9** 重复**步骤8**，产生data_block设置次数的128bits随机数据。查询HTRNG_STATUS.test_ready，为1表示第一次Generate子流程计算完成。标准的NIST SP 800-90A DRBG known-answer测试丢弃第一次Generate功能的结果。设置HTEST_DRBG_CTRL.test_drbg_gen=0。
- 步骤10** 写入384bit的向量（第二次Reseed的EntropyInput和AdditionInput的异或值），然后设置HTEST_DRBG_CTRL.test_drbg_reseed=1，开始进行DRBG的第二次reseed子流程计算。
- 步骤11** 查询HTRNG_STATUS.test_ready，为1表示第二次reseed子流程计算完成。设置HTEST_DRBG_CTRL.test_drbg_reseed=0，设置HTEST_DRBG_CTRL.test_drbg_gen=1。开始进行DRBG的第二次Generate子流程计算。
- 步骤12** 查询HTRNG_INT_STATUS.ready_int_status，为1表示128bits随机数据产生，设置HTRNG_INT_SOURCE.ready_int_source=1，清除随机数据ready中断状态位。
- 步骤13** 重复**步骤12**，产生data_block设置次数的128bits随机数据。查询HTRNG_STATUS.test_ready，为1表示第二次Generate子流程计算完成。查询HTRNG_STATUS.test_ready，为1表示第二次Generate子流程计算完成。
- 步骤14** 清除HTRNG_CONTROL.drbg_enable0，然后清除HTRNG_TEST.test_mode，HTEST_DRBG_CTRL.test_drbg_init，HTEST_DRBG_CTRL.test_drbg_gen，HTEST_DRBG_CTRL.test_drbg_reseed，HTRNG_MODE.trng_mode退出测试模式。

----结束

下面是一个针对CTR_DRBG测试向量



图 9-14 CTR_DRBG 测试向量

EntropyInput =
c54805274bde00aa5289e0513579019707666d2fa7a1c890
8865891c87c0c652335a4d3cc415bc30742b164647f8820f

PersonalizationString = d63fb5afa2101fa4b8a6c3b89d9c250ac728fc1ddad0e7585
b5d54728ed20c2f940e89155596e3b963635b6d6088164b

TRNG_DRBG_PS_AI_0 = 0x47f8820f XOR 0x6088464b

⋮

TRNG_DRBG_PS_AI_11 = 0xc5480527 XOR 0xd63fb5af

第一次写入

图 9-15 第二次写入

AdditionalInput = 744bfae3c23a5cc9a3b373b6c50795068d35eb8a339746ac
810d16f864e880061082edf9d2687c211960aa83400f85f9

EntropyInputPR = b2ad31d1f20dcf30dd526ec9156c07f270216bdb59197325
bab180675929888ab699c54fb21819b7d921d6346bff2f7f

TRNG_DRBG_PS_AI_0 = 0x400f85f9 XOR 0x6bff2f7f

⋮

TRNG_DRBG_PS_AI_11 = 0x744bfae3 XOR 0xb2ad31d1

第二次写入

图 9-16 第三次写入

AdditionalInput = ad55c682962aa4fe9ebc227c9402e79b0aa7874844d33eae
e7e2d15baf81d9d33936e4d93f28ad109657b512aee115a5

EntropyInputPR = eca449048d26fd38f8ca435237dce66eadec7069ee5dd0b70
084b819a711c0820a7556bbd0ae20f06e5169278b593b71

TRNG_DRBG_PS_AI_0 = 0xae115a5 XOR 0x8b593b71

⋮

TRNG_DRBG_PS_AI_11 = 0xad55c682 XOR 0xeca44904

第三次写入



图 9-17 结果数据

ReturnedBits =
f08dfc1775b6feb5a4177110bf29d7c3ab715dfdc4b27200359288c0624bd5c1028acc9914d88a82b09f5
eaafdc3bca8547b98481df39b86504314221cbdc3c 生成的512bits DRBG数据

9.6.2.7 获取噪声源数据操作流程

在FIPS的相关随机数认证中需要对噪声源产生的随机数据进行评估。噪声源数据直接来自于FROs，没有经过后处理。具体的操作步骤如下：

- 步骤1 设置寄存器HTRNG_CONTROL，写全0。配置HTRNG_INT_SOURCE为0xFFF，清除起始的无效中断。确保HTRNG处在一个确定状态。
- 步骤2 设置寄存器HTRNG_SAMPLE_CFG.sample_div，设置从FROs采样的时钟间隔，0表示每个时钟都从FROs采样，15表示每16个时钟周期采样一次。设置HTRNG_SAMPLE_CFG.sample_cycles，设置异或到一起生成1bit的FRO采样bit数目，异或结果再移位到主移位寄存器中。如果测试到的噪声源数据，随机效果不理想，则增大sample_div和sample_cycles的值，有益于提高熵值。
- 步骤3 配置寄存器HTRNG_NOISE_ENABLE，使能噪声源振荡环输出。
- 步骤4 配置寄存器HTRNG_TEST.test_mode=1使能测试模式。设置HTRNG_TEST.test_shfreq=1，使能噪声源数据寄存器测试有效。然后配置寄存器HTRNG_CONTROL.trng_enable=1，使能HTRNG。
- 步骤5 置位HTRNG_TEST.test_noise_vld=1，使能测试模式下噪声源的采样。
- 步骤6 查询HTRNG_STATUS.test_noise_ready，为1表示64bits噪声源数据采样完毕。读取HTRNG_NOISE_L/H。
- 步骤7 重复步骤5到步骤6，直到采样到需要的足够数据。
- 步骤8 清除HTRNG_TEST.test_mode，HTRNG_TEST.test_shfreq，退出测试模式。

----结束

9.7 性能

9.7.1 实际测试结果

9.7.2 吞吐率（Throughput）

表 9-2 DRBG 随机数产生吞吐率

行为	-
RNDR获取随机数	16B/16cycles*Fclk（1.5G）1.5GB/s



9.7.3 延迟（ Latency ）

表 9-3 DRBG 随机数产生延迟

行为	延迟（ Latency ） Value
产生一个data block（ 128bit ）	16Cycles
Reseed	169cycles
熵源产生（ 8*512bit噪声产生 384bits全熵 ）	4096cycles

9.8 DFX 应用指导

9.8.1 模块维测

9.8.1.1 中断上报

TRNG的中断包括：

- 128bit随机数ready中断。
- FROs错误关断中断。
- DRBG连续产生2个128bits相同随机数错误中断。
- AIS噪声源检测失效中断（ noise_fail ）。
- AIS游程检测失败中断（ run_fail ）。
- AIS长游程检测失败中断（ long_run_fail ）。
- AIS扑克检测失败中断（ poker_fail ）。
- AIS长游程检测失败中断（ long_run_fail ）。
- AIS单比特频数检测失败中断（ monobit_fail ）。
- NRBG连续产生2个相同随机数错误中断（ stuck_nrbg ）。
- 重复计数检测错误中断（ repcnt_fail ）。
- 自适应均衡检测错误中断（ aprop_fail ）。

软件检测到噪声源异常中断之后，可以通过重新启动FROs（ 将 HTRNG_NOISE_ENABLE.gring_en， HTRNG_NOISE_ENABLE.fring_en， HTRNG_NOISE_ENABLE.ring_en寄存器先关闭再打开 ）的方式来对噪声源进行调整。然后清除中断。

9.8.1.2 消息上报

无

9.8.1.3 Debug

无



9.9 注意事项

NA